出版说明

MATLAB 是当今最优秀的科技应用软件之一，它以强大的科学计算与可视化功能、简单易用、开放式可扩展环境，特别是所附带的 30 多种面向不同领域的工具箱支持，使得它在许多科学领域中成为计算机辅助设计和分析、算法研究和应用开发的基本工具和首选平台。

MATLAB 具有其他高级语言难以比拟的一些优点，如编写简单、编程效率高、易学易懂等，因此 MATLAB 语言也被通俗地称为演算纸式科学算法语言。在控制、通信、信号处理及科学计算等领域中，MATLAB 都被广泛地应用，已经被认可为能够有效提高工作效率、改善设计手段的工具软件，掌握了 MATLAB 就好比掌握了开启这些专业领域大门的钥匙。

MATLAB 是从事众多工业、科研领域的必备工具。无论是在校学生，还是已经参加工作的工程技术人员和科研人员，都非常渴望快速学习 MATLAB 并熟练运用它来解决各种科学问题、工程问题。非常遗憾的是，目前市场上很难找到一套能够从入门到精通快速掌握该软件的最新学习资料，致使使用者在学习中遇到了实际问题而难以解决。虽然 MATLAB 软件本身具有一定的帮助功能，但是它们阅读起来并不方便，而且某些重要的概念没有给予详细的解释和说明，应用举例也偏少，使用者难以快速掌握它。

这套丛书的推出，将在 MATLAB 新版本软件和使用者之间架起一座桥梁，让国内的工程技术人员无需花费太多的时间和精力，就能尽快掌握该软件及它的一些新特性和新功能，并通过大量的实例告诉使用者如何解决面临的实际问题。

本套丛书首批将推出 5 种图书，简介如下：

MATLAB 7 基础与提高
全面系统地介绍了 MATLAB 7 这个功能强大的软件。首先详细讲解了 MATLAB 数值运算、符号运算、程序设计初步和基本绘图功能；然后举出了很多应用实例，旨在通过实践操作巩固学习前面所介绍的知识；最后讲述了 MATLAB 的高级部分，包括 GUI 界面设计、Simulink、Notbook、几种常用的工具箱，以及外部程序接口知识等。
小波分析理论与 MATLAB 7 实现
以最新推出的小波分析工具箱 Wavelet Toolbox 3.0 版本为基础。全书共分为三大部分，第 1 部分着重介绍了小波理论基础，包括小波基础知识、连续小波变换、离散小波变换、多分辨率分析与正交小波变换、小波变换和多采样滤波器组、二维小波变换与图像处理及小波包的基本原理等；第 2 部分重点说明了小波分析工具箱的详细使用方法，包括图形用户接口、小波通用函数、一维小波变换的 MATLAB 实现、二维小波变换的 MATLAB 实现、小波包变换的 MATLAB 实现、信号和图像的降噪和压缩，以及最新的信号和图像的提升小波变换等内容；第 3 部分主要介绍了小波工具箱的应用基础，以及小波变换在语音和生物医学信号处理中、故障诊断中、数字水印中的应用方法。

MATLAB 7 辅助控制系统设计与仿真
通过介绍 MATLAB 7 软件及其控制系统工具箱的使用方法，并结合控制系统的设计流程及实际应用，全面系统地介绍了控制系统设计与仿真的全过程。全书内容由浅入深，以工程应用为背景，从基础知识、建模与分析、设计与仿真流程三个方面对控制系统的设计与仿真进行了深入的说明，同时书中列举大量实例，尽量贴近工程实际，具有很强的代表性。
MATLAB 7 辅助信号处理技术与应用
系统地介绍了信号与系统基础知识、常用信号变换、离散系统结构、IIR 数字滤波器设计、FIR 数字滤波器设计、平稳信号分析、非平稳信号分析、高斯信号分析及信号处理的 GUI 实现。其中，信号与系统基础知识包括连续信号与模型、离散信号与模型；常用信号变换包括 z 变换、Chirp z 变换、FFT 变换、DCT 变换和 Hilbert 变换等；离散系统结构包括 IIR、FIR 和 Lattice 结构；IIR 滤波器设计包括模拟和数字低通、高通、带通与带阻滤波器设计，以及基于冲激响应不变法和双线性 z 变换法的 IIR 滤波器设计等；FIR 滤波器设计包括基于窗函数、频率抽样法和切比雪夫逼近法的 FIR 滤波器设计；平稳信号分析包括经典功率谱估计、基于参数模型的功率谱估计和基于非参数模型的功率谱估计；非平稳信号分析包括 STFT 变换、Gabor 展开、Wigner-Ville 分布与 Choi-Williams 分布；非高斯信号分析包括基于非参数法的双谱估计、基于参数模型的双谱估计，以及双谱估计的应用；信号处理的 GUI 实现包括滤波器设计与分析的 FDATool 工具和滤波器设计与信号分析的 SPTool 工具。
神经网络理论与 MATLAB 7 实现
以最新推出的神经网络工具箱 4.0.3 版本为基础。本书前两章介绍了 MATLAB 7 和神经网络的基础知识，对神经网络工具箱的重要函数分门别类地进行了详细介绍，并给出了完整的示例。从第 3 章到第 5 章，分别介绍了几种比较重要的神经网络类型，包括感知器、线性网络和 BP 网络等，并介绍了这些网络的结构及学习算法，以及 MATLAB 的实现方法。第 6 章介绍了神经网络的图形用户界面。后 5 章分别讲述了如何利用神经网络工具箱解决控制、故障诊断、预测和有源消声等应用领域中的实际问题。

总之，这套书涵盖了MATLAB使用基础、高级编程和重要领域的应用，相信这套丛书的推出，将为MATLAB工程技术人员提供最权威最系统的知识参考，帮助他们快速解决学习、科研和工程实际中面临的问题。

我们的联系方式如下：
咨询电话：（010）68134545　　68131648
电子邮件：support@fecit.com.cn
服务网址：http://www.fecit.com.cn　　http://www.fecit.net
通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

飞思科技产品研发中心

前　言

自上世纪80年代末以来，神经网络这个涉及多种学科的新的高科技领域，吸引了众多的神经生理学家、心理学家、数理科学家、计算机与信息科学家及工程师和企业家等进行研究和应用。大量的有关神经网络机理、模型、算法特性分析，以及在各领域应用的学术论文像雨后春笋般在报刊杂志上和许多国际学术会议中涌现。神经网络日益成为当代高科技领域中方兴未艾的竞争热点。

MATLAB 是一款强大的工程计算和仿真软件，刚刚发布的 R14 产品族比以往任何版本都更加强大，其中的神经网络功能提供了大量可直接调用的函数和命令，基本上囊括了目前应用比较成熟的神经网络设计方法。用 MATLAB 来编写各种网络设计与训练的子程序，可以使用户从繁琐的编程中解脱出来，大大提高工作效率和解题质量。因此，如何应用神经网络工具箱函数来解决工程实践中的问题已成为燃眉之急。我们根据自身多年来从事神经网络系统设计和 MATLAB 使用的经验编写了本书。除了详细介绍各个工具箱函数之外，还着重讲解了利用 MATLAB 进行神经网络系统分析与设计的大量实例。

本书是介绍应用 MATLAB 软件 R14 版本进行神经网络设计和应用的最新书籍。本书前两章介绍了 MATLAB 7 和神经网络的基础知识，对神经网络工具箱的重要函数分门别类地进行了详细介绍，并给出了完整的示例。从第3章到第5章，分别介绍了几种比较重要的神经网络类型，包括感知器、线性网络和 BP 网络等，并介绍了这些网络的结构及学习算法，以及 MATLAB 的实现方法。第6章介绍了神经网络的图形用户界面。后5章分别讲述了如何利用神经网络工具箱解决控制、故障诊断、预测和有源消声等应用领域中的实际问题。

全书通过大量的 MATLAB 实例为读者讲述了神经网络的 MATLAB 实现方法，形象生动，图文并茂，深入浅出，脉络清晰，可读性强。相信广大读者通过认真学习本书，可以快速学会神经网络技术和 MATLAB 实现方法，并建立牢固的知识基础，真正做到“事半功倍”，起到课堂上无法达到的学习效果。

本书由飞思科技产品研发中心策划并组织编写，孙志强、葛哲学负责全书的统稿与审校工作。孙志强、廖剑利、张建、肖俊同志负责本书第1章到第5章的编写；葛哲学、邱忠、安卫华、杨勇等负责本书第6、7、9、10章的编写；刘瑛、李浩明、潘薇、陈仲生负责本书的第8章和第11章的编写。此外，张丽娜、安莹、孙金华、刘美琴、张珏琼、谢光军、朱国强、郭玉玲、卿慧玲、王勇、葛诚、胡雷、胡艳等负责书稿的材料整理和测试工作，并提供了大量的帮助与意见。另外，还有很多同志在本书的编校过程中付出了大量的劳动，在此一并表示衷心的感谢。

本书可作为各领域工程技术人员的参考用书，也可作为高等学校理工类各专业高年级本科生和研究生的神经网络课程的教材，还可作为其他科技工作者应用神经网络的参考资料。

由于时间仓促加之作者本身水平有限，书中错误之处在所难免。在此，敬请各领域专

家和广大读者批评指正。

我们的联系方式如下：
咨询电话：（010）68134545　　68131648
电子邮件：support@fecit.com.cn
服务网址：http://www.fecit.com.cn　　http://www.fecit.net
通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

编著者
2005年1月1日

目 录

第 1 章　概　　述

本章主要介绍了 MATLAB 和神经网络的一些基础知识。首先介绍了 MATLAB 7 及其神经网络工具箱的知识，这里介绍的知识在以后的几章中都会用到，是基于 MATLAB 7 进行神经网络分析、设计与实现工作的知识基础。对于神经网络，本章从神经网络的产生、发展，神经网络模型的建立，以及人工神经网络等方面进行了概括性的说明，使读者对神经网络有个初步的认识。

本章主要内容：

- MATLAB 语言简介
- MATLAB 快速入门
- 神经网络发展史

1.1　MATLAB 语言简介

1.1.1　MATLAB 概述

MATLAB 诞生于 20 世纪 70 年代，它的编写者是 Cleve Moler 博士和他的同事。当时，他们利用 Fortran 开发了两个子程序库——EISPACK 和 LINPACK。这两个子程序库是求解线性方程的程序库。但是，Cleve Moler 发现学生使用这两个程序库有困难，主要问题是因为接口程序不好写，很费时间。于是，Cleve Moler 自己动手，在业余时间里编写了 EISPACK 和 LINPACK 的接口程序。Cleve Moler 给这个接口程序取名为 MATLAB，意为矩阵（Matrix）和实验室（Laboratory）的组合。以后几年，MATLAB 作为免费软件在大学里被广泛使用，深受大学生的喜爱。

1984 年，Cleve Moler 和 John Little 成立了 MathWorks 公司，正式把 MATLAB 推向市场，并继续进行 MATLAB 的开发。1993 年，MathWorks 公司推出 MATLAB 4.0；1995 年，MathWorks 公司推出 MATLAB 4.2C 版（For Win3.*x*）；1997 年，推出 MATLAB 5.0；2000 年 10 月，MathWorks 公司推出 MATLAB 6.0；2002 年 8 月，发布了 MATLAB 6.5，2004 年 9 月发布了最新版本的 MATLAB 7。每一次版本的推出都使 MATLAB 有了长足的进步，界面越来越友好，内容越来越丰富，功能越来越强大，帮助系统越来越完善。

MATLAB 长于数值计算，能处理大量的数据，而且效率比较高。MathWorks 公司在此基础上加强了 MATLAB 的符号计算、文字处理、可视化建模和实时控制能力，增强了 MATLAB 的市场竞争力，使 MATLAB 成为了市场主流的数值计算软件。

MATLAB 产品族是支持从概念设计、算法开发、建模仿真到实时实现的理想的集成环境。无论是进行科学研究还是产品开发，MATLAB 产品族都是必不可少的工具。

1．MATLAB 的应用领域

MATLAB 产品族可用于以下领域：

- 数据分析
- 数值和符号计算
- 工程与科学绘图
- 控制系统设计
- 数字图像信号处理
- 财务工作
- 建模、仿真、原型开发
- 图形用户界面设计等

MATLAB 产品族被广泛地应用于包括信号与图像处理、控制系统设计、系统仿真等诸多领域。开放式的结构使 MATLAB 产品族很容易针对特定的需求进行扩充，从而在不断深化对问题的认识的同时，提高自身的竞争力。

MATLAB 产品族的一大特性是有众多的面向具体应用的工具箱和仿真模块，包含了完整的函数集，用来对信号与图像处理、控制系统设计、神经网络等特殊应用进行分析和设计。同时，其他的产品也延伸了 MATLAB 的能力，包括数据采集、报告生成和依靠 MATLAB 语言编程产生独立的 C/C++代码等。

2．MATLAB 的产品构成

MATLAB 主要产品有：

- MATLAB：所有 MathWorks 公司产品的数值分析和图形基础环境。MATLAB 将 2D 和 3D 图形、MATLAB 语言编程集成到一个单一的、易学易用的环境之中。
- MATLAB Toolbox：一系列专用的 MATLAB 函数库，用于解决特定领域的问题。工具箱是开放的、可扩展的，可以查看其中的算法或开发自己的算法。
- MATLAB Compiler：将 MATLAB 语言编写的 M 文件自动转换成 C 或 C++文件，支持用户进行独立的应用开发。结合 Mathworks 提供的 C/C++数学库和图形库，用户可以利用 MATLAB 快速地开发出功能强大的独立应用系统。
- Simulink：是结合了框图界面和交互仿真能力的非线性动态系统仿真工具。它以 MATLAB 的数学、图形和语言为基础。
- Stateflow：与 Simulink 框图模型相结合，描述复杂事件驱动系统的逻辑行为，驱动系统可以在不同的模式之间进行切换。
- Real-Time Workshop：直接从 Simulink 框图自动生成 C 或 Ada 代码，用于实现快速原型和硬件的仿真，整个代码的生成可以根据需要完全定制。
- Simulink Blockset：专门为特定领域设计的 Simulink 功能模块的集合，用户也可以利用已有的模块或自行编写的 C 和 MATLAB 程序建立自己的模块。

3．MATLAB 的功能

MATLAB 的核心是一个基于矩阵运算的快速解释程序，它交互式地接收用户输入的各项命令，输出计算结果。MATLAB 提供了一个开放式的集成环境，用户可以运行系统提供的大量命令，包括数值计算、图形绘制和代码编制等。具体来说，MATLAB 具有以下功能：



- 数据可视化功能
- 矩阵运算功能
- 大量的工具箱
- 绘图功能
- GUI 设计
- Simulink 仿真

通过运用 MATLAB 这些强大的功能，工程师、科研人员、数学家和教育工作者可以在统一的平台下完成相应的科学计算工作。这些工作涉及到的领域非常广泛，例如汽车、电子、仪器仪表和通讯等。

MathWorks 公司刚刚推出了 MATLAB R14 版本产品，即 MATLAB 7，主要包括 12 个新产品模块，同时还升级了 28 个产品模块。

1.1.2 MATLAB 语言特点

MATLAB 语言有不同于其他高级语言的特点，它被称为第四代计算机语言。如同第三代计算机语言（如 Fortran 语言、C 语言等）使人们摆脱了对计算机硬件的依赖一样，第四代语言 MATLAB 使人们从繁琐的程序代码中解放了出来。它丰富的函数库使开发者省去了大量的重复编程。MATLAB 语言最大的特点就是简单和快捷。

1. 编程效率高

MATLAB 是一种面向科学与工程计算的高级语言，允许用数学形式的语言来编写程序，比 Basic、Fortran 和 C 等语言更加接近我们书写计算公式的思维方式，用 MATLAB 编写程序犹如在演算纸上排列出公式与求解问题一样。因此，MATLAB 语言也可以通俗地称为“演算纸”式科学算法语言，正是由于它编写简单，所以编程效率高，易学易懂。

2. 用户使用方便

MATLAB 语言是一种解释执行的语言（在没被专门的工具编译之前），它灵活、方便，调试手段丰富，调试速度快。人们用任何一种语言编写程序和调试程序一般都要经过 4 个步骤：编辑、编译、连接，以及执行和调试。各个步骤之间是顺序关系，编程的过程就是在它们之间做瀑布型的循环。MATLAB 语言与其他语言相比，把编辑、编译、连接和执行融为一体。它能在同一界面上进行灵活操作，快速排除输入程序中的书写错误、语法错误，甚至语义错误，从而加快了开发者编写、修改和调试程序的速度，可以说，在编程和调试过程中它是一种比 Visual Basic 还要简单的语言。

具体地说，在运行 MATLAB 时，如果直接在命令行输入 MATLAB 语句（命令），包括调用 M 文件的语句，每输入一条语句，就会立即对其进行处理，完成编译、连接和运行的全过程。又如，将 MATLAB 源程序编辑为 M 文件时，由于 MATLAB 磁盘文件也是 M 文件，所以编辑后的源文件就可直接运行，而不需要进行编译和连接。在运行 M 文件时，如果有错误，计算机屏幕上就会给出详细的出错信息，使它经过修改后再执行，直到正确为止。所以，MATLAB 语言不仅是一种语言，广义上更可以称为一种语言开发系统、语言调试系统。

3．扩充能力强，交互性好

高版本的 MATLAB 语言拥有丰富的库函数，在进行复杂的数学运算时可以直接调用，而且MATLAB的库函数同用户文件在形成方式上一样，所以用户文件也可以作为MATLAB的库函数被调用。因而，开发者可以根据自己的需要方便地建立和扩充新的库函数，以便提高 MATLAB 的使用效率和扩充它的功能。另外，为了充分利用 Fortran、C 等语言的资源，包括开发者已经编辑好的 Fortran、C 语言等程序，可以通过建立 Me 文件的形式，进行混合编程，方便地调用有关的子程序；还可以在 C 语言和 Fortran 语言中方便地使用 MATLAB 的数值计算功能，这些良好的交互性使程序员可以使用以前编写过的程序，减少重复性工作，也使编写的程序具有可重复利用的价值。

4．移植性很好，开放性很好

MATLAB 是用 C 语言编写的，而 C 语言的可移植性很好。于是，MATLAB 可以很方便地被移植到能运行 C 语言的操作平台上。MATLAB 适合的工作平台有 Windows 系列、UNIX、Linux、VMS 6.1 和 PowerMac。除了内部函数外，MATLAB 所有的核心文件和工具箱文件都是公开的，都是可读可写的源文件，用户可以通过对源文件的修改和自己编程构成新的工具箱。

5．语句简单，内涵丰富

MATLAB 语言中最基本最重要的成分是函数，其一般形式为[a,6,c,…]= fun(d,e,f,…)，即一个函数由函数名、输入变量 d,e,f,…和输出变量 a,b,c,…组成。同一函数名 F、不同数目的输入变量（包括无输入变量）及不同数目的输出变量，代表着不同的含义（有点像面向对象中的多态性）。这不仅使 MATLAB 的库函数功能更丰富，而且大大减少了需要的磁盘空间，使得 MATLAB 编写的 M 文件简单、短小而高效。

6．高效方便的矩阵和数组运算

MATLAB 语言像 Basic、Fortran 和 C 语言一样规定了矩阵的算术运算符、关系运算符、逻辑运算符、条件运算符及赋值运算符，而且这些运算符大部分可以原封不动地照搬到数组间的运算中，有些如算术运算符只要增加“.”就可用于数组间的运算。另外，它不需要定义数组的维数，只需给出矩阵函数、特殊矩阵专门的库函数，使之在求解诸如信号处理、建模、系统识别、控制、优化等领域的问题时，显得更加简捷、高效，这是其他高级语言所不能比拟的。在此基础上，高版本的 MATLAB 已逐步扩展到科学及工程计算的其他领域。因此，不久的将来，它一定能名副其实地成为“万能演算纸”式的科学算法语言。

7．方便的绘图功能

MATLAB 的绘图功能是十分方便的，它有一系列绘图函数（命令），例如线性坐标、对数坐标、半对数坐标及极坐标。使用时均只需调用不同的绘图函数（命令），在图上标出图题、*XY* 轴标注，格（栅）绘制也只需调用相应的命令，简单易行。另外，在调用绘图函数时可以通过调整自变量绘出不变颜色的点、线、复线或多重线。这种为科学研究着想的设计是其他通用的编程语言所不及的。



1.1.3 MATLAB 7 的安装

MATLAB 7 支持的操作系统平台有：

- Windows 2000（SP3 或 SP4）
- Windows NT 4.0（SP5 或 SP6）
- Windows XP
- Linux ix86 2.4.*x*，glibc 2.2.5
- Sun Solaris 2.8，2.9
- HPUX 11.0 和 11.1
- Mac OS X 10.3.2

MATLAB 7 的安装程序与旧版本相比有新的特点，包括 Typical（典型）和 Custom（自定义）安装选项及新的放置文档的方法。

（1）Typical 和 Custom 安装方法

Windows 平台上的 MATLAB 7 安装程序提供了两种安装路径，即“典型”安装和“自定义”安装。

Typical 安装是一种最简单的安装方式，这种方式将安装所有得到许可的产品，安装过程不会显示让用户进行选择的产品的清单，一般情况下所有产品会安装在一个默认的目录下，用户也可以自行更改安装目录。

Custom 安装方式使得用户可以选择自己想要安装的产品，而且可以设置其他几种安装选项，这是 Typical 安装方式下所没有的。比如，用户可以选定所有的安装文件为只读文件。

（2）在线帮助文档的安装

MATLAB 7 安装程序经常会自动安装每一个产品的在线 HTML 文档。由于在线文档以 JAR 文件的压缩格式存储，因此它比以前的版本占用更少的磁盘存储空间。另外，MATLAB 7 的安装盘中没有 PDF 文档，用户可以到该站点上下载：http://www.mathworks.com/access/helpdesk/help/helpdesk.html。

1.1.4 MATLAB 7 的新特点

2004 年，MathWorks 公司发布了 MATLAB 的最新版本产品 MATLAB 7。MATLAB 7 的特点在于其具有全新的桌面及各种不同领域的集成工具，更易于用户使用。多种新工具简化了许多一般的工作，如资料输入、快速分析，创造高品质且具实用性的图表分析等。MATLAB 7 在编程和代码效率、绘图和可视化、数学运算、数据读写等方面有了很大改进。其新的特色和功能加强包括以下 3 个方面。

（1）开发环境

- 可以重新设计桌面，更容易管理多个文档、图形，能够保存自定义的窗口布局，以及设置常用命令的快捷键；
- 增强了矩阵编辑器和工作空间浏览器，更容易察看、编辑和显示变量；
- M-Lint 代码检验器对代码进行修改，以使性能和可维护性最优；

- 能够在编辑器中执行 M 代码的个别部分。

（2）编程

- 能够生成嵌套函数，使得定义和调用自定义函数更容易；
- 提供了匿名函数，允许在命令行或脚本语言中定义单行函数；
- 能够使用标准的语法来调用函数句柄，而不是 feval；
- 设置条件断点，可以在任何表达式为真（True）时停止代码运行。

（3）数学运算

- 单精度的数学运算、线性代数、FFT 和滤波函数，使得 MATLAB 可以处理更大的单精度数据集；
- “鲁棒”的计算几何路径（使用 Qhull 2002.1），更能控制算法选择；
- linsolve 函数使得求解线性方程更快；
- ODE 求解器能处理隐性微分方程及多点边界值问题。

（4）绘图功能

- 新的绘图接口，可以在不进入 M 代码下交互式地生成和编辑图形；
- 从图形产生 M 代码，可以重用自定义的绘图；
- 增强的图形注释，包括图形形状，对象排列等；
- 能够进行各种图形变换，比如旋转、平移等。

（5）数据读写和外部接口

- 新的文件输入/输出函数，能够读非常大的任意格式的文本文件，并能写入 Excel 和 HDF5 文件；
- 压缩的 MAT 文件选项，可以使用更少的磁盘空间来存储大的数据集；
- javaaddpath 函数可以在不重新启动 MATLAB 下动态地添加、移出和重新载入 Java 类；
- FTP 对象可以连接 FTP 服务器，进行远程数据操作；
- 支持 COM 自定义接口、服务器事件和 Visual Basic 脚本。

（6）性能和平台支持

- JIT 加速器涵盖了所有的数据类型和函数调用；
- JIT 加速器产生 MMX 函数，可用于整数运算；
- 使用新版本的 FFTW3 实现 FFT；
- 更快的文本文件读写操作。

1.1.5 MATLAB 7 的新产品及更新产品

1．新产品

MATLAB 7 在保留原有产品的基础上，增加了许多新的产品。

（1）MATLAB 新产品

- MATLAB Builder for COM
- MATLAB Builder for Excel

（2）新的工具箱

- Bioinformatics Toolbox（生物信息科学工具箱）

  生物信息科学工具箱提供了一个集成的软件计算环境，用于分析基因和蛋白质，主要特色包括：

  a）能够连接网络数据库；

  b）可以读和转化多种格式的数据文件；

  c）能够计算数据的统计特性；

  d）能够利用隐马尔科夫模型对生物信号序列的统计模式进行建模。

- Filter Design HDL Coder（滤波器设计 HDL 代码生成器）

  滤波器设计 HDL 代码生成器可以利用 MATLAB 和滤波器工具箱来设计固定点的滤波器，它能生成高效的 VHDL 或 Verilog 代码，其测试平台用于快速仿真、测试和检验产生的滤波器代码。

- Fixed-point Toolbox（固定点工具箱）

  固定点工具箱为 MATLAB 提供了固定点的数据类型，使其能够开发固定点的各种算法，它允许生成以下类型的对象：

  a）fi：定义一个固定点的数值对象；

  b）fimath：定义 fi 对象的数学属性；

  c）fipref：定义 fi 对象的现实属性；

  d）numerictype：定义 fi 对象的数据类型和尺度属性；

  e）quantizer：量化数据集。

- Genetic Algorithm and Direct Search Toolbox（遗传算法和直接搜索工具箱）

  遗传算法和直接搜索工具箱包含了一系列的函数，它扩展了优化工具箱和 MATLAB 数值计算环境的能力，使其可以通过以下方式来求解优化问题：

  a）遗传算法；

  b）直接搜索。

- Link for ModelSim

  Link for ModelSim 是一个仿真接口，它将 MathWorks 的工具与 EDA 集成起来，可用于可编程门阵列和 ASIC 电路设计。这个接口提供了 Mentor 图形化的 HDL 仿真器、ModelSim SE/PE 和 MATLAB 及 Simulink 之间快速的双向连接，可以进行直接的硬件设计、验证和仿真。

- OPC Toolbox

  OPC Toolbox 包含了一系列的基于 MATLAB 计算环境的 M 文件函数和 MEX 文件动态连接库，它主要有以下特点：

  a）提供了一个能与一个或多个 OPC 服务器交互的框架；

  b）与 OPC 基本的数据接入标准兼容；

  c）与 OPC 服务器进行事件驱动的交互；

  d）面向对象的层次结构用于管理与 OPC 服务器的连接。

- RF Toolbox

  RF 工具箱可以生成和集成 RF 电路，用于在频域对功率和噪声进行仿真，而

且还可以读、写、分析、组合和可视化 RF 网络参数。

（3）Simulink 新产品

- Embedded Target for TI C2000 DSP Platform

  Embedded Target for TI C2000 DSP Platform 1.0 是一个新的产品，可以生成、仿真和下载可执行代码到 C2000 DSP 目标板上。

- Simulink Control Design（Simulink 控制设计）

  Simulink Control Design 提供了对 Simulink 中的控制系统和物理模型进行线性化的工具，它使用了图形用户接口使得设置运行条件、模型线性化和分析结果更加简单。线性化后的 Simulink 模型使得系统分析和补偿器设计变得更为方便，它的应用领域包括：

  a）航空：飞行控制、导航；

  b）自动化领域：巡航控制、发射控制；

  c）设备制造：电机、磁盘驱动。

- Simulink Parameter Estimation（Simulink 参数估计）

  Simulink Parameter Estimation 可以利用经验输入/输出数据来估计一个 Simulink 模型的参数和初始状态，它提供了一个图形用户接口使得上述估计过程更加简单。

（4）新的模块

- RF Blockset（RF 模块）

  RF Blockset 是一个设计、分析和仿真 RF 通信系统的工具，它在时域中使用基带同一行为模型来建模和分析 RF 系统，RF 模块可以让开发者使用 RF 元件库来组合复杂的 RF 系统。

- Signal Processing Blockset（信号处理模块）

  Signal Processing Blockset 是原产品 DSP Blockset 的改名。

2. 更新的产品

MATLAB 7 在增加新产品的同时，还对原有的产品进行了更新。

（1）MATLAB 产品更新

- MATLAB Complier
- MATLAB Report Generator
- Excel Link

（2）工具箱更新

- Communications Toolbox（通信工具箱）
- Control System Toolbox（控制系统工具箱）
- Data Acquisition Toolbox（数据采集工具箱）
- Database Toolbox（数据库工具箱）
- Filter Design Toolbox（滤波器设计工具箱）
- Image Processing Toolbox（图像处理工具箱）
- Instrument Control Toolbox（仪器控制工具箱）



- Optimization Toolbox（优化工具箱）
- Signal Processing Toolbox（信号处理工具箱）
- Virtual Reality Toolbox（虚拟现实工具箱）
- System Identification Toolbox（系统辨识工具箱）
- Wavelet Toolbox（小波工具箱）

（3）Simulink 更新

- Embedded Target for Infineon® C166 Microcontrollers
- Embedded Target for Motorola® HC12
- Embedded Target for Motorola® MPC555
- Embedded Target for OSEK/VDX®
- Embedded Target for TI C6000™ DSP
- Real-Time Windows Target
- Real-Time Workshop®
- SimMechanics
- SimPowerSystems
- xPC Target
- xPC TargetBox™

（4）模块更新

- Communications Blockset
- Dials&Gauges Blockset
- DSP Blockset
- Fix-Point Blockset
- Nonlinear Control Design Blockset
- Signal Processing Blockset

1.1.6 Simulink 6.0 的新特点

Simulink 是一个多领域仿真和基于模型的动态系统设计的平台，它提供了一个交互式的图形环境和一个自定义的模块库集，使得开发者可以准确地进行系统的设计、仿真、应用、测试和控制。Simulink 6.0 改进了响应率、建模逼真度和工作效率，其新特点主要包括以下 4 个方面。

（1）基于元件的大系统建模

- 能够将一个模型分段成多个文件，每一个文件是一个分离的模型；
- 能够在集成系统模型之前，对每一个单独的元件进行建模、仿真、测试和应用；
- 增强了将用户模型集成到基于已有文件的配置管理和版本控制软件的功能；
- 增加了大模型的更新图表和仿真的速度；
- 提供了分开的工作空间，用于每个模型参数和变量的存储和管理；
- 增强了用于定义接口的总线支持。

（2）Simulink 和状态流集成

- 统一的模型搜索器，用于导航、产生、配置和搜索用户模型的所有信号、参数和属性；
- 统一的仿真和代码产生选项；
- 产生和存储多个仿真和代码生成配置。

（3）数据管理和可视化

- 新的数据对象用于定义结构、总线及自定义的数据类型对象；
- 在不增加模块到模型的情况下可以增加测试点；
- 信号和示波器管理器能够在不增加模块的情况下将 Source 和 Sinks 连接到模型。

（4）MATLAB 语言支持

- 从嵌入式的 MATLAB 算法可以生成 C 代码；
- 增强了生成 M 文件的 S 函数的功能。

1.2 MATLAB 快速入门

自 MATLAB 6.x 上市以来，相对于以前的版本，MATLAB 最突出和最实用的地方是向用户提供了前所未有的成系列的交互式工作界面。了解、熟悉和掌握这些交互界面的基本功能和操作方式，对于正确快速地利用神经网络工具箱可以起到事半功倍的作用。因此，本节将专门介绍 MATLAB 主窗口中最常用的交互界面：历史指令窗、当前目录浏览器、工作空间浏览器、内存数组编辑器、交互界面分类目录窗、M 文件编辑/调试器及帮助导航/浏览器。

1.2.1 命令行窗口

命令行窗口（Command Window）如图 1-1 所示。可以在命令行窗口中输入 MATLAB 命令，可以是一个单独的 MATLAB 语句，也可以是一段利用 MATLAB 编程功能实现的代码。

例 1.1 在命令行窗口创建一个 BP 网络。在命令行窗口中输入：

```
net = newff([0 10],[5 1],{'tansig' 'purelin'});
```

输入完成后按回车键，这条命令就会被执行，执行结果如图 1-1 所示。图中是对刚刚创建的 BP 网络的详细介绍。

在创建神经网络时，经常需要用到矩阵，在 MATLAB 命令行中，矩阵有两种输入方式。

例 1.2 输入一个简单的矩阵$\begin{bmatrix} 0 & 1 \\ 0 & 2 \end{bmatrix}$，该矩阵为神经网络输入向量元素的范围。

方法 1：在命令行中输入 A=[0 1;0 2]；

方法 2：在命令行中输入 A=[0 1
　　　　　　　　　　　　0 2]。

按回车键后，输出结果都是

```
A =
     0     1
```

```
        0    2
```

如果需要获取矩阵 A 中某个元素，只需在命令行窗口中输入 A(i,j)，即可获得矩阵 A 的第 i 行第 j 列的元素。如

```
A(2,1)=
     0
```

若需要获取矩阵的某行向量，只需输入 A(i,:)，其中 i 为所需的行数。同理，A(:,j)可以获得矩阵的第 j 列向量。如

```
A (2,:)=
     0    2
```

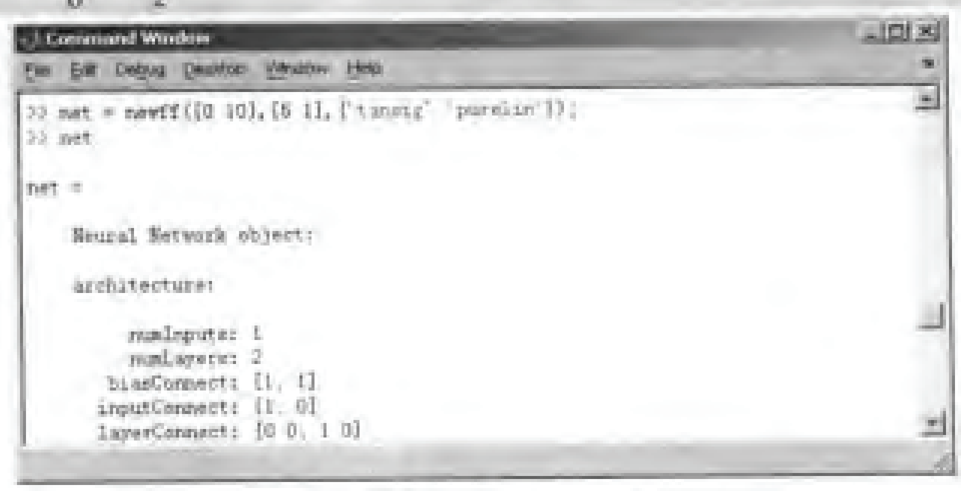


图 1-1 命令行窗口

在利用 MATLAB 进行编程或者在命令行中输入命令时，都有可能出现命令或代码比较长的情况，这里就需要使用命令的续行输入。

例 1.3 在命令行窗口中续行输入矩阵 A=[1 2 3 4 5 6; 1 2 3 5 7 8]。

输入命令为：

```
A=[1 2 3 4 5 6;
   1 2 3 5 7 8]
```

回车后输出结果为：

```
A =
     1    2    3    4    5    6
     1    2    3    5    7    8
```

例 1.4 在命令行窗口中利用 plot 函数绘制正弦函数图形。

函数 plot 在利用神经网络工具箱时会被经常用到，它是最基本的二维绘图函数。plot 的基本调用格式有以下 3 种。

（1）plot(x)

如果 x 为向量，则以 x 元素值为纵坐标，以相应元素的下标作为横坐标来绘函数图。如果 x 为实数矩阵，则按列绘制每列元素值相对其下标的连线图，图中曲线等于 x 阵的列数。如果 x 为复数矩阵，则分别以 x 的实部阵和虚部阵对应的列元素为横纵坐标绘制多条连线图。

（2）plot(x,y)

如果 x、y 为同维向量，则绘制以 x、y 为横纵坐标的连线图。如果 x 是向量，y 是一

个与 x 同维的矩阵，则绘制多条不同色彩的连线图，连线条数等于 y 阵的另一维数。如果 x 和 y 是同维矩阵，则以 x、y 对应元素为横纵坐标分别绘制曲线，曲线的条数等于矩阵的行数。

（3）plot(x,y,s)

s 表示线条的颜色和类型，如 s='r+'，表示各点是由红色的+号绘制的。如果没有特别注明，默认的类型为蓝色的线条。

现有一个正弦函数 y=sin(x)，绘制其函数图。在命令行窗口中输入以下代码后按回车键：

```
x=-5:0.1:5;
y=sin(x);
plot(x,y,'r');
```

绘制结果如图 1-2 所示。

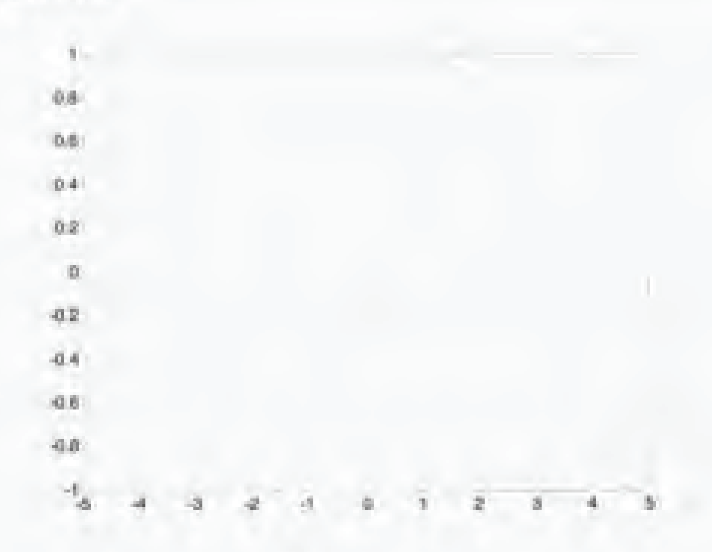


图 1-2　正弦函数

对于较复杂的绘图过程，还需要其他比较重要的函数和标记，应用比较多的有 hold on、hold off 和 figure。前两者用于保存绘制句柄，适用于在同一张图上绘制多条曲线的情形；后者用于停止绘制句柄，表示重开一张图进行绘制。

接下来绘制三条正弦函数曲线，其中在第一张图上绘制两条，在第二张图上绘制一条。在命令行窗口中输入以下代码后按回车键：

```
x=-5:0.1:5;
y1=sin(x);
y2=sin(2*x);
y3=sin(1.5*x);
plot(x,y1);
hold on
plot(x,y2,'r+');
hold off
figure;
plot(x,y3,'bo');
```

绘制结果如图 1-3 和图 1-4 所示。

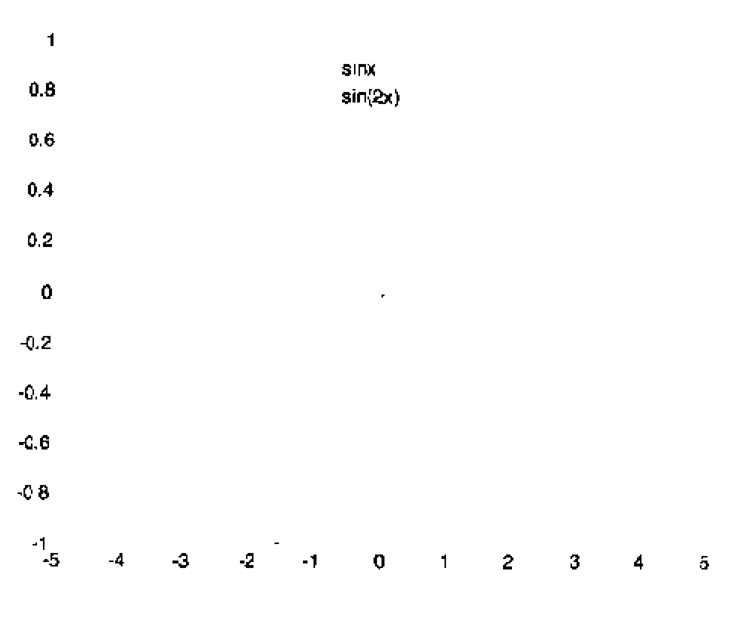


图 1-3　函数 y=sinx 和 y=sin(2x)

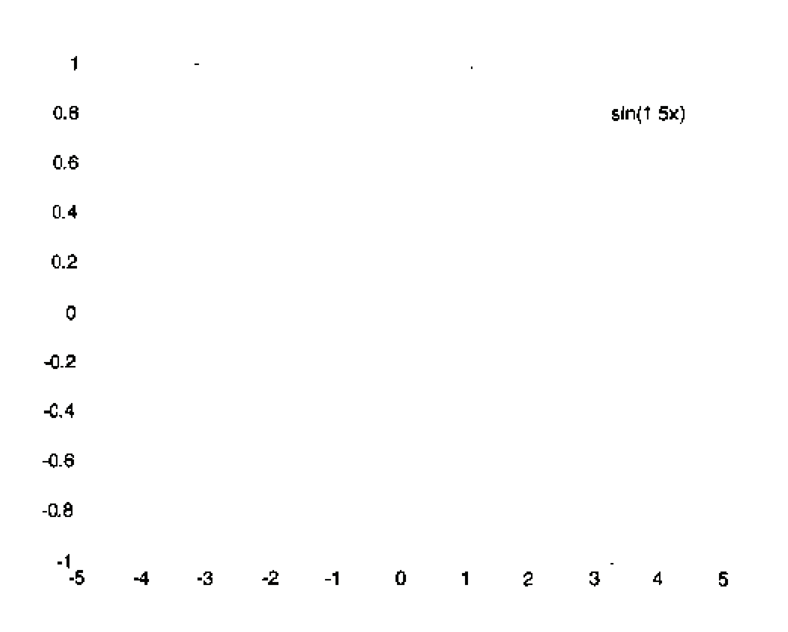


图 1-4　函数 y=sin(1.5x)

1.2.2　其他重要窗口

MATLAB 主窗口中还有几个比较重要的窗口，如命令行历史窗口（Command History）、当前路径窗口（Current Directory）等。这些窗口和命令行窗口是密切相关的，它们的设置会影响到命令行窗口中命令的执行情况。

1. Command History 窗口

该窗口中存储了在命令行窗口中输入的所有命令，如图 1-5 所示。如果想重新执行已经运行过的命令，该窗口提供了一个很好的选择。

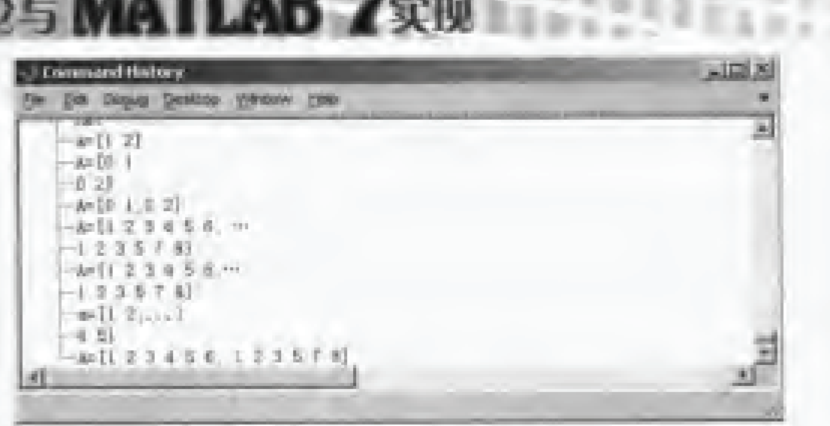


图 1-5 Command History 窗口

如果需要重新运行该窗口中的命令，只需在窗口中选中该命令，然后双击鼠标即可。若需要重新运行多行命令，只需按住【Shift】键，然后选中需要重新执行的命令，双击选中的命令就行了。

2. Current Directory 窗口

该窗口指定了当前的，如果不加以改动的话，每次启动 MATLAB，该窗口默认的当前为 X:/MATLAB 7/work（X 为 MATLAB 的安装盘符），如图 1-6 所示。

该窗口显示了当前下所有的文件和文件夹，双击和 MATLAB 相关的文件即可运行之。单击按钮，就可以改变当前的路径。



如图 1-6 所示的 M 文件 bsb.m，当前路径下的文件在当前路径发生改变的情况下，就无法在命令行窗口中正常运行了。会出现如下的错误提示：

??? Undefined function or variable 'bsb'



图 1-6 Current Directory 窗口

3. 工作空间浏览器（Workspace Browser）

工作空间浏览器存储并显示了当前命令行窗口中所有的变量，如图 1-7 所示。这些变量是保存在内存中的，在 MATLAB 进程结束以前，一直是活动的。

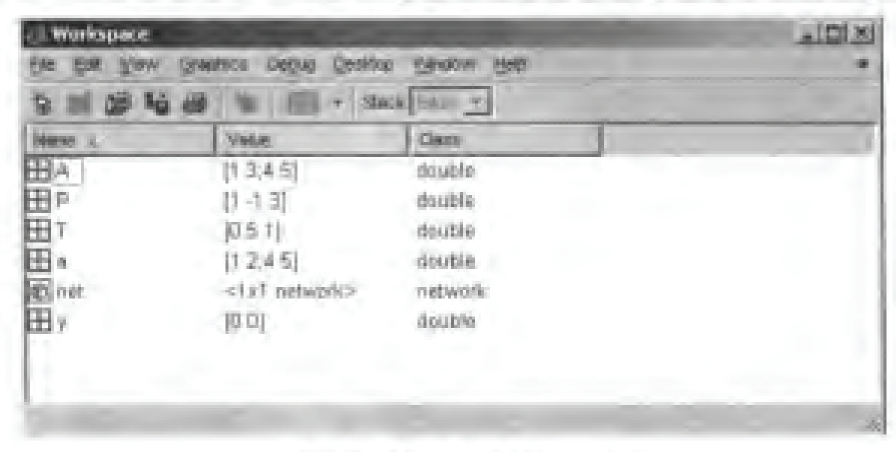


图 1-7 Workspace Browser

在命令行窗口中输入 who 和 whos，可以查看当前内存中的所有变量，包括变量的名称、大小和类型。

```
>> who
Your variables are:
A    P    T    a    net  y

>> whos
  Name      Size          Bytes   Class
  A         2×2           32      double array
  P         1×2           16      double array
  T         1×2           16      double array
  a         2×2           32      double array
  net       1×1           18903   network object
  y         1×2           16      double array
Grand total is 648 elements using 19015 bytes
```

在命令行窗口中输入 clear，可以清除当前内存中的所有变量。

```
>> clear
```

此时，再次输入 who，已经没有任何变量显示了。查看 Workspace Browser 窗口，其中变量显示部分也变成了空白。

1.2.3 Editor/Debugger 窗口

该窗口用于 MATLAB 脚本的编写和执行，如图 1-8 所示。脚本的编写有很多注意事项，许多 MATLAB 书籍和资料已经给出了大量的说明，这也不是本书的重点，因此，就不多做介绍了。

当脚本编写结束后，保存为一个 M 文件，此时，可以从【Debug】菜单中选择【Run】命令，也可以直接按【F5】键来执行该文件。

另外，也可以在命令行窗口中输入文件名后按回车键来运行该文件。当然，前提是编写的 M 文件没有错误。

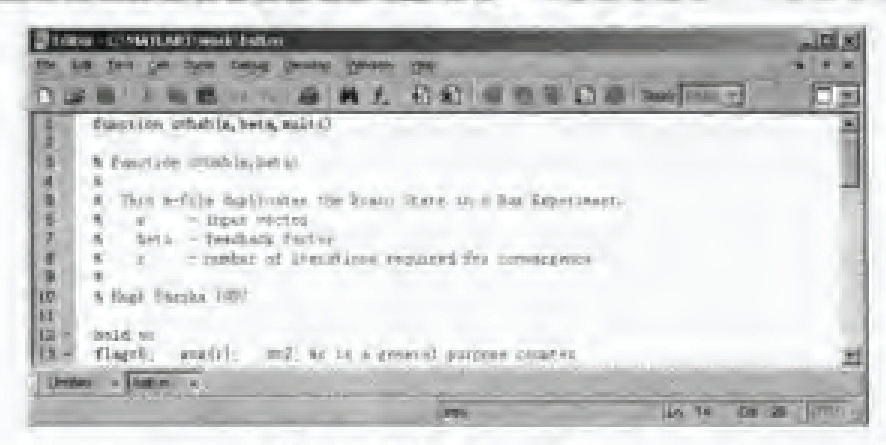


图 1-8　Editor/Debugger 窗口



不要采用纯数字作为 M 文件的名字，这样会导致错误的结果。如果有一个名为 1.m 的 M 文件，运行后的结果只能是 ans=1。

1.2.4　MATLAB 帮助系统

MATLAB 7 提供了丰富的帮助资源，有 PDF 形式，也有网页形式。所有的 PDF 帮助文档都在安装过程中默认放置在 X:/MATLAB 7/help/pdf_doc 文件夹中，读者可以选择需要的帮助文档进行查看。

在命令行窗口中输入 help 命令后按回车键，显示了所有的帮助主题，结果为：

```
HELP topics
MATLAB\general        -  General purpose commands.
MATLAB\ops            -  Operators and special characters.
MATLAB\lang           -  Programming language constructs.
MATLAB\elmat          -  Elementary matrices and matrix manipulation.
MATLAB\elfun          -  Elementary math functions.
......
```

如果需要某个函数或工具箱的帮助信息，可以在命令行窗口中输入如下命令：

```
>>help name
```

其中，name 为需要帮助的函数或工具箱的名称。例如，需要有关前向型网络的创建函数 newff 的有关信息，可在命令行窗口中输入如下命令并按回车键：

```
>>help newff
```

结果为：

```
NEWFF Create a feed-forward backpropagation network.
    Syntax
     net = newff
     net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
......
```

结果中提供了丰富的帮助信息，包括函数的功能、调用方法、各参数的意义、使用过程中的注意事项和示例，最后还给出了与该函数相关的其他函数的名称。

另外，在 MATLAB 主窗口中，在【Help】菜单下选中【MATLAB Help】命令，就可

以启动集成式的帮助系统，如图 1-9 所示。这个帮助系统非常完善，而且容易查询，可以轻松地找出所需的帮助信息。图中显示了神经网络工具箱中前向型网络创建函数 newff 的帮助信息。

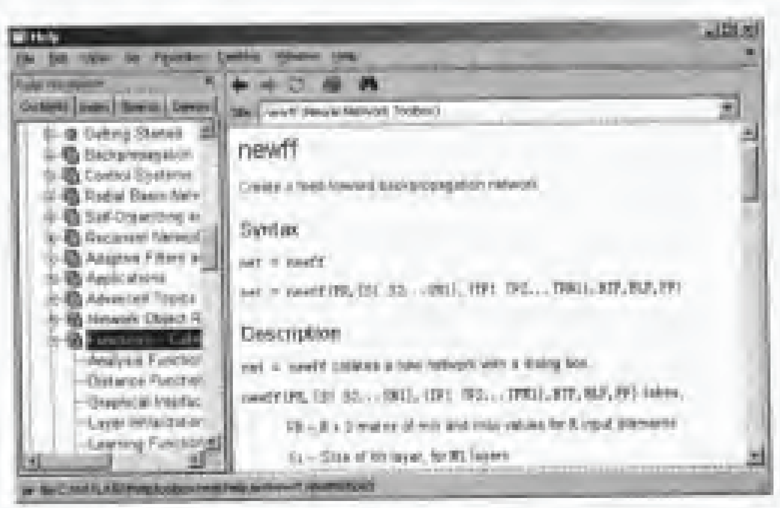


图 1-9 集成式帮助系统

1.2.5 神经网络工具箱快速入门

MATLAB 7 对应的神经网络工具箱的版本号为 Version 4.0.3，它以神经网络理论为基础，利用 MATLAB 脚本语言构造出典型神经网络的激活函数，如线性、竞争性和饱和线性等激活函数，使设计者对所选定网络输出的计算，变成对激活函数的调用。另外，根据各种典型的修正网络权值的规则，再加上网络的训练过程，利用 MATLAB 编写出各种网络设计和训练的子程序，网络设计人员可以根据自己的需要去调用工具箱中有关的设计和训练程序，将自己从繁琐的编程中解脱出来，集中精力解决其他问题，从而提高了工作效率。

最新版本的神经网络工具箱几乎涵盖了所有的神经网络的基本常用模型，如感知器和 BP 网络等。对于各种不同的网络模型，神经网络工具箱集成了多种学习算法，为用户提供了极大的方便。另外，工具箱中还给出了大量的示例程序和帮助文档，能够快速地帮助用户掌握工具箱的应用方法。

目前，神经网络工具箱中提供的神经网络模型主要应用于：

- 函数逼近和模型拟合；
- 信息处理和预测；
- 神经网络控制；
- 故障诊断。

在实际应用中，面对一个具体的问题时，首先需要分析利用神经网络求解问题的性质，然后依据问题特点，确定网络模型。最后通过对网络进行训练、仿真等，检验网络的性能是否满足要求。下面介绍具体的过程。

1. 确定信息表达方式

将领域问题及其相应的领域知识转化为网络可以接受并处理的形式，即将领域问题抽

象为适合于网络求解所能接受的某种数据形式。尽管在实际应用中，问题的形式会是多种多样的，但不外乎以下几种：

- 数据样本已知；
- 数据样本之间相互关系不明确；
- 输入/输出模式为连续的或者离散的；
- 输入数据按照模式进行分类，模式可能会具有平移、旋转或伸缩等变化形式；
- 数据样本的预处理；
- 将数据样本分为训练样本和测试样本。

2. 网络模型的确定

主要是如何根据问题的实际情况，选择模型的类型、结构等。另外，还可在典型网络模型的基础上，结合问题的具体情况，对原网络进行变形、扩充等，同时还可以采用多种网络模型的组合形式。

3. 网络参数的选择

确定网络的输入输出神经元的数目，如果是多层网络，还需要进一步确定隐含层神经元的个数。对于反馈神经网络，如 Hopfield 网络和 Elman 网络，还需要进一步地设置反馈神经元的有关属性。

4. 训练模式的确定

包括选择合理的训练算法，确定合适的训练步数，指定适当的训练目标误差，以获得较好的网络性能。

5. 网络测试

选择合理的测试样本，对网络进行测试，或者将网络应用于实际问题，检验网络性能。值得指出的是，网络测试过程需要遵循“交叉测试”的原则，保证测试的有效性、准确性和全面性。

总之，神经网络工具箱是一个内容全面、操作方便的软件包，对于广大神经网络系统的研发者来说，掌握神经网络工具箱的应用将使得自己在工作中如虎添翼。

1.3 神经网络发展史

神经网络系统理论的发展历史是不平衡的，自 1943 年心理学家 McCulloch 和数学家 Pitts 提出神经元生物学模型（简称 M-P 模型）以来，至今已经有 50 多年的历史了。在这 50 多年的发展历史中，大体可以分为以下几个发展阶段。

1.3.1 初期阶段

自 1943 年 M-P 模型开始，至该世纪 60 年代为止，这一段时间可以称为神经网络系统理论发展的初期阶段，这个时期的主要特点是多种网络的模型的产生与学习算法的确定，如 1944 年 Hebb 提出了 Hebb 学习规则，该规则至今仍是神经网络学习算法的一个基本规



则；1957 年 Rosenblatt 提出了感知器（Perceptron）模型；1962 年 Widrow 提出了自适应（Adaline）线性元件模型等。这些模型和算法在很大程度上丰富了神经网络系统理论。

1.3.2 停滞期

上个世纪 60 年代到 70 年代，神经网络系统理论的发展处于一个低潮时期，造成这种情况的原因是发展过程中遇到了本质的困难，即电子线路交叉极限的困难（对 n 个神经元就存在 n^2 条连线）。在当时的条件下，神经元数量 n 的大小受到极大的限制，因此神经网络系统不可能完成高度集成化、智能化的计算任务。同时，神经网络系统理论本身也有很多不完善的地方。所以，神经网络系统理论与应用研究工作进展缓慢。另一方面，这一时期正是数字计算机发展的全盛时期，无论在硬件、软件还是技术应用和商品市场方面都取得了突飞猛进的发展，使得大批有才华的科学家的注意力都转移到数值计算机方面了。

虽然形势如此严峻，但仍有很多科学家在困难条件下坚持开展研究，并提出了很多种不同的网络模型，展开了增加网络功能和改善学习算法等方面的研究，为神经网络系统发展的高潮奠定了坚实的基础。

Stephen Grossberg 是这些人中最有影响力的，他深入研究了心理学和生物学的处理，以及人类信息处理的现象，把思维和脑紧密地结合在一起，成为了统一的理论。

芬兰的 Kohonen 在 1971 年开始了随机连接变化表方面的研究工作，从次年开始，他将研究目标集中到联想记忆方面。Kohonen 将 LVQ 网络应用到语音识别、模式识别和图像识别方面，取得了很大的成功。

1.3.3 黄金时期

从 20 世纪 80 年代开始，是神经网络系统理论发展的黄金时期。这个时期最具标志性的人物是美国加州工学院的物理学家 John Hopfield。他于 1982 年和 1984 年在美国科学院院刊上发表了两篇文章，提出了模仿人脑的神经网络模型，即著名的 Hopfield 模型。Hopfield 网络是一个互连的非线性动力学网络，它解决问题的方法是一种反复运算的动态过程，这是符号逻辑处理方法所不具备的性质。

20 世纪 80 年代，关于智能计算机发展道路的问题日趋迫切地提高到日程上来，由于计算机的集成度日趋极限状态，但数值计算的智能水平与人脑相比，仍有较大的差距。因此，就需要从新的角度来思考智能计算机的发展道路问题。这样一来，神经网络系统理论重新受到重视。所以，20 世纪 80 年代后期到 90 年代初，神经网络系统理论形成了发展的热点，多种模型、算法和应用问题被提出，研究经费重新变得充足，完成了很多有意义的工作。

目前，神经网络系统理论与技术的发展大体分为以下三个方面进行。

首先在硬件技术方面，一些发达国家，如美国和日本均实现了规模超过 1000 个神经元的网络系统，这样的系统具有极高的运算速度，而且已经在股票数据分析中得到了应用。另外，为了克服电子线路交叉极限问题，很多国家都在研究电子元件之外的神经网络系统，如光电子元件和生物元件等。

在神经网络系统理论的研究方面，主要进展有 Boltzmann 机理论的研究、细胞网络的提出和性能指标的分析等。

神经网络系统的应用研究主要集中在模式识别（语音和图像识别）、经济管理和优化控制等方面，它和数学、统计中的多个学习有着密切的联系，如线性和非线性规划问题、数值逼近、统计计算等。另外，在其他信息处理问题中也有很多应用，如数据压缩、编码、密码和股市分析等领域，应用内容十分丰富。

1.3.4 发展展望

20 世纪 90 年代中期是神经网络系统理论进行稳健发展的时期，在经历了 20 世纪 80 年代末与 90 年代初的发展高潮之后，人们肯定了它的前途，但同时又看到了它发展的障碍。与 20 世纪 60 年代到 70 年代相比，80 年代到 90 年代的发展无论是在硬件技术还是在应用范围和理论水平方面的贡献都是巨大的。但是神经网络系统的基本困难，即电子线路交叉的困难和理论研究问题的困难仍然没有实现根本性的突破。按照目前的集成电路水平，已经实现了 1000 个神经元的网络，这样的规模已经很可观了，但是与人体具有的神经元数目动辄 10^{10}～10^{15} 相比，仍然还有较大的差距。因此，如何克服网络连线困难仍是神经网络技术发展过程中需要克服的最关键的问题。

另外，在神经网络系统理论研究方面，还有许多问题尚待解决，如按照生物测试，每个神经元只有 10^4～10^5 个突触，这些神经元是如何连接，又是如何工作的？这一问题还有待于生物、医学和数学工作者的研究才可能解决。另外，对现有的神经网络系统，也有许多问题，如多层感知器的学习算法问题、Hopfield 网络的假吸引点问题、大量工程应用中提出的神经网络模型中的学习算法问题，都迫切需要解决。

1.4 神经网络模型

神经网络是由大量的处理单元（神经元）互相连接而成的网络。为了模拟大脑的基本特性，在神经科学研究的基础上，提出了神经网络的模型。但是，实际上神经网络并没有完全反映大脑的功能，只是对生物神经网络进行某种抽象、简化和模拟。神经网络的信息处理通过神经元的相互作用来实现，知识与信息的存储表现为网络元件互连分布式的物理联系。神经网络的学习和识别取决于各神经元连接权系数的动态演化过程。

1.4.1 神经元结构模型

神经元是神经网络的基本处理单元，一般表现为一个多输入、单输出的非线性器件，通用的结构模型如图 1-10 所示。

其中，u_i 为神经元 i 的内部状态，θ_i 为阈值，x_j 为输入信号，w_{ij} 表示与神经元 x_j 连接的权值，s_i 表示某一外部输入的控制信号。



$$\begin{cases}\tau\dfrac{\mathrm{d}u_i}{\mathrm{d}t}=-u_i(t)+\sum w_{ij}x_j(t)-\theta_i\\ y_i(t)=f\left[u_i(t)\right]\end{cases}$$

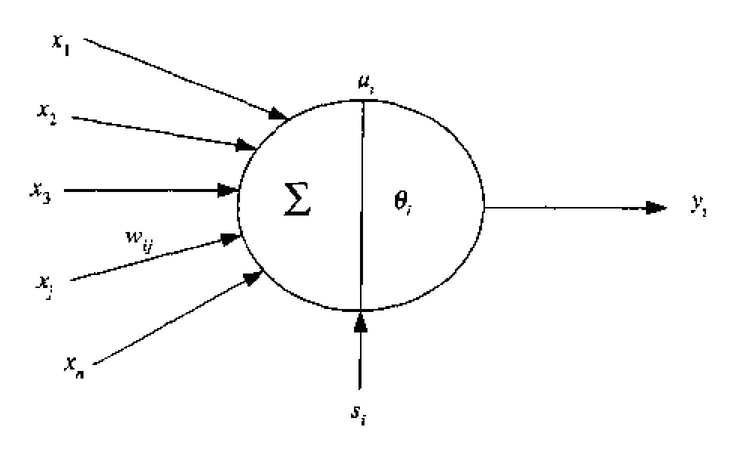


图 1-10　神经元结构模型

神经元模型常用一阶微分方程来描述，它可以模拟生物神经网络突触膜电位随时间变化的规律。

神经元的输出由函数 f 表示，一般利用以下函数表达式来表现网络的非线性特征。

（1）阈值型，为阶跃函数

$$f(u_i)=\begin{cases}1 & u_i\geqslant 0\\ 0 & u_i<0\end{cases}$$

（2）分段线性型

$$f(u_i)=\begin{cases}1 & u_i\geqslant u_2\\ au_i+b & u_i\leqslant 0<u_2\\ 0 & u_i<u_1\end{cases}$$

（3）S 型函数

$$f(u_i)=\frac{1}{1+\exp(-u_i/c)^2}$$

其中，c 为常数。

S 型函数反映了神经元的饱和特性，由于其函数连续可导，调节曲线的参数可以得到类似阈值函数的功能，因此，该函数被广泛应用于许多神经元的输出特性中。

1.4.2　神经网络的互连模式

根据连接方式的不同，神经网络的神经元之间的连接有如下几种形式。

1. 前向网络

前向网络结构如图 1-11 所示，神经元分层排列，分别组成输入层、中间层（也称为隐含层，可以由若干层组成）和输出层。每一层的神经元只接受来自前一层神经元的输入，后面的层对前面层没有信号反馈。输入模式经过各层次的顺序传播，最后在输出层上得到输出。感知器网络和 BP 网络均属于前向网络。

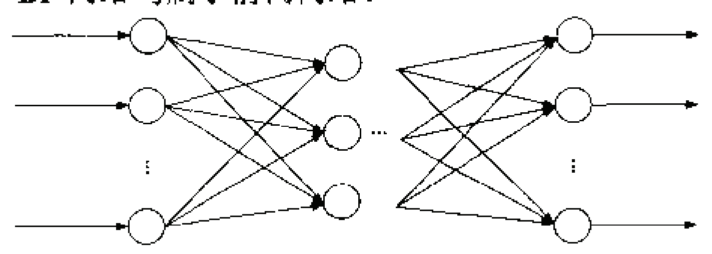


图 1-11　前向网络结构

2. 有反馈的前向网络

其结构如图 1-12 所示，从输出层对输入层有信息反馈，这种网络可用于存储某种模式序列，如神经认知机和回归 BP 网络都属于这种类型。

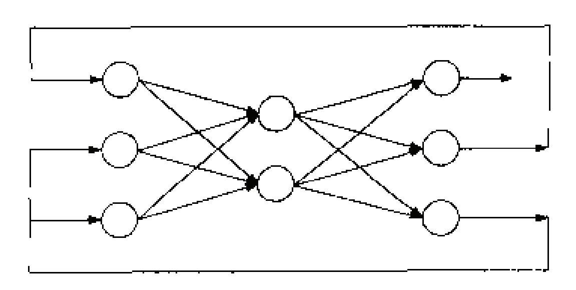


图 1-12　有反馈的前向网络结构

3. 层内有相互结合的前向网络

其结构如图 1-13 所示，通过层内神经元的相互结合，可以实现同一层内神经元之间的横向抑制或兴奋机制。这样可以限制每层内可以同时动作的神经元素，或者把每层内的神经元分为若干组，让每一组作为一个整体进行运作。例如，可利用横向抑制机理把某层内具有最大输出的神经元挑选出来，从而抑制其他神经元，使之处于无输出的状态。

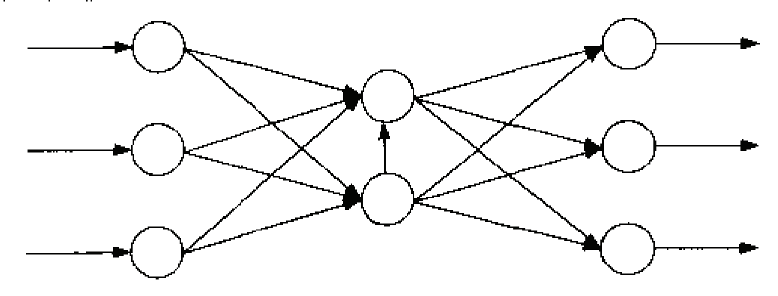


图 1-13　层内有相互结合的前向网络结构

4. 相互结合型网络（全互连或部分互连）

相互结合型网络结构如图 1-14 所示，这种网络在任意两个神经元之间都可能有连接。Hopfield 网络和 Boltzmann 机均属于这种类型。在无反馈的前向网络中，信号一旦通过某

神经元，该神经元的处理就结束了。而在相互结合网络中，信号要在神经元之间的反复传递，网络处于一种不断改变状态的动态之中。信号从某初始状态开始，经过若干次变化，才会达到某种平衡状态。根据网络的结构和神经元的特性，网络的运行还有可能进入周期振荡或其他如混沌等平衡状态。

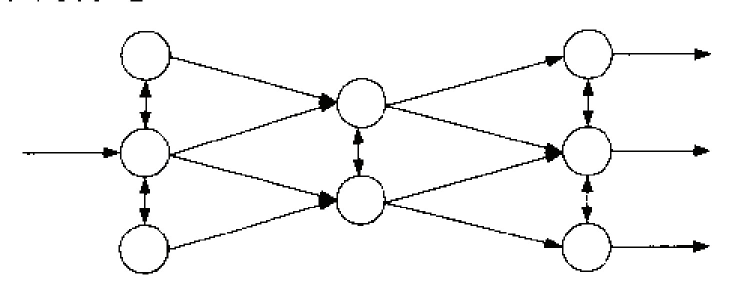


图 1-14　相互结合型网络结构

1.5　神经网络的特性及实现

神经网络的全称是人工神经网络（Artificial Neural Network，ANN），它采用物理上可实现的器件或采用计算机来模拟生物体中神经网络的某些结构和功能，并应用于工程领域。神经网络的着眼点不在于利用物理器件完整地复制生物体中的神经细胞网络，而是抽取其中可利用的部分来克服目前计算机或其他系统不能解决的问题，如学习、控制、识别和专家系统等。随着生物和认知科学的发展，人们对大脑的认识和了解越来越深入，神经网络必然会获得更加广阔的发展空间和应用范围。

虽然 ANN 与真正的生物神经网络有差别，但由于它汲取了生物神经网络的部分优点，因此具有一些固有的特性。

（1） ANN 在结构上与目前的计算机本质不同，它是由很多小的处理单元互相连接而成的，每个处理单元的功能简单，但大量简单的处理单元集体的、并行的活动得到预期的识别、计算的结果，具有较快的速度。

（2）ANN 具有非常强的容错性，即局部的或部分的神经元损坏后，不会对全局的活动造成很大的影响。

（3） ANN 记忆的信息是存储在神经元之间的连接权值上，从单个权值中看不出存储信息的内容，因而是分布式的存储方式。

（4） ANN 的学习功能十分强大，它的连接权值和连接的结构都可以通过学习得到。

1.6　小　　结

本章对 MATLAB 的产生背景和发展过程、神经网络工具箱的特性，以及一些简单的使用方法、MATLAB 7 的最新特性进行了简单介绍，并概述了神经网络的发展历史、分类、特性和实现等。可以说，神经网络具有强大的容错功能，可以比较轻松地实现非线性映射过程，并且具有大规模计算的能力。因此，它在自动化、计算机和人工智能领域都有着广

泛的适用性，实际上也确实得到了大量的应用，解决了很多利用传统方法难以解决的问题。而 MATLAB 神经网络工具箱的出现，更加拓宽了神经网络的应用空间。神经网络工具箱将很多原本需要手动计算的工作交给计算机，一方面提高了工作效率；另一方面，还提高了计算的准确度和精度，减轻了工程人员的负担。

接下来我们正式开始学习神经网络工具箱。

第2章 神经网络工具箱函数及实例

神经网络理论是在20世纪提出的，自此以后，由于自身固有的超强适应能力和学习能力，神经网络在很多领域获得了极其广泛的应用，解决了许多传统方法难以解决的问题，发挥了巨大的作用。迄今为止，神经网络在自动控制、计算机科学、机器学习、故障诊断与检测、心理学乃至经济学等领域都有着大量的应用。

虽然神经网络有着广泛的实用性和强大的解决问题的能力，但是它也存在一些缺陷。比如，神经网络的建立实际上就是一个不断尝试的过程，以 BP 网络为例，网络的层数及每一层结点的个数都是需要不断地尝试来改进的。同样，对于神经网络的学习过程来说，固然已经有很多已经成形的学习算法，但这些算法在数学计算上都比较复杂，过程也比较繁琐，容易出错。因此，采用计算机辅助进行神经网络设计与分析就成为了必然的选择。

目前，已经有一些比较成熟的神经网络软件包。其中，应用最为广泛的软件包之一就是 MATLAB 的神经网络工具箱。自从 MATLAB 5.*x* 提供了该工具箱后，它已经成为工程人员进行神经网络分析与设计的首选。该工具箱随着 MATLAB 版本的发展，自身也在不断完善与提高，现在最新的为 4.0.3 版本，是迄今为止最为全面和强大的。

本章概要介绍了最新版的 MATLAB 神经网络工具箱的各种网络的函数。

本章主要内容有：

- 神经网络工具箱中的公共函数
- 感知器的神经网络工具箱函数
- BP 网络的神经网络工具箱函数
- 线性网络的神经网络工具箱函数
- 自组织竞争网络的神经网络工具箱函数
- 径向基网络的神经网络工具箱函数
- 反馈网络的神经网络工具箱函数

2.1 概 述

神经网络工具箱是在 MATLAB 环境下开发出来的许多工具箱之一。它以人工神经网络理论为基础，利用 MATLAB 编程语言构造出许多典型神经网络的激活函数，如 S 型、线性、竞争层、饱和线性等激活函数，使设计者对所选定网络输出的计算，转变为对激活函数的调用。另外，可以根据各种典型的修正网络权值的规则，加上网络的训练过程，利用 MATLAB 语言编写各种网络权值训练的子程序。这样一来，网络的设计者可以根据自己的需要调用工具箱中有关神经网络的设计与训练的程序，使自己能够从繁琐的编程中解脱出来，集中精力思考和解决问题，从而提高效率和质量。

MATLAB 7 所对应的神经网络工具箱 V4.0.3 的内容非常丰富，包含了很多现有的神经

网络新成果，其中涉及到以下几种网络模型：

- 感知器模型
- 线性滤波器
- BP 网络模型
- 控制系统网络模型
- 径向基网络模型
- 自组织网络
- 反馈网络
- 自适应滤波和自适应训练

另外，该工具箱中还包括大量的演示实例，有助于初学者加深理解。它所介绍的网络模型和实例可以通过 MATLAB 的帮助进行学习，前提是英语水平要足够高。如在 MATLAB 的命令窗口中键入 help nnet，即可得到神经网络工具箱的有关版本信息及函数列表，如：

```
>> help nnet
  Neural Network Toolbox
  Version 4.0.3 (R14) 05-May-2004   ——版本信息
```

以下为函数列表：

```
Graphical user interface functions.——GUI 函数
    nntool     - Neural Network Toolbox graphical user interface.

  Analysis functions.——分析函数
    errsurf   - Error surface of single input neuron.
    maxlinlr - Maximum learning rate for a linear layer.
......
```

限于篇幅，此处只节选了部分结果，其中中文为作者添加的。

2.2 神经网络工具箱中的通用函数

神经网络工具箱中准备了丰富的工具函数。其中一些函数是特别针对某一种类型的神经网络的，如感知器的创建函数、BP 网络的训练函数等；而另外一些函数则是通用的，几乎可以用于所有类型的神经网络，如神经网络仿真函数、初始化函数和训练函数等。

本节主要介绍了通用的神经网络工具函数，对它们的功能、调用格式、使用方法及注意事项做了详尽的说明。表 2-1 列出了神经网络中一些比较重要的通用函数。

表 2-1　通用函数

函数类别	函数名称	函数用途
神经网络仿真函数	sim	针对给定的输入，得到网络输出
神经网络训练函数	train	调用其他训练函数，对网络进行训练
	trainb	对权值和阈值进行训练
	adapt	自适应函数
神经网络学习函数	learnp	网络权值和阈值的学习

（续表）

函数类别	函数名称	函数用途
初始化函数	learnpn	标准学习函数
	revert	将权值和阈值恢复到最后一次初始化时的值
初始化函数	init	对网络进行初始化
	initlay	多层网络的初始化
	initnw	利用 Nguyen-Widrow 准则对层进行初始化
	initwb	调用指定的函数对层进行初始化
神经网络输入函数	netsum	输入求和函数
	netprod	输入求积函数
	concur	使权值向量和阈值向量的结构一致
传递函数	harlim	硬限幅函数
	hardlims	对称硬限幅函数
其他	dotprod	权值求积函数

2.2.1 神经网络仿真函数 sim

该函数用于对神经网络进行仿真，调用格式为：

```
[Y,Pf,Af,E,perf] = sim(net,P,Pi,Ai,T)
[Y,Pf,Af,E,perf] = sim(net,{Q TS},Pi,Ai,T)
[Y,Pf,Af,E,perf] = sim(net,Q,Pi,Ai,T)
```

其中，

Y：函数返回值，网络输出；

Pf：函数返回值，最终输出延迟；

Af：函数返回值，最终的层延迟；

E：函数返回值，网络误差；

perf：函数返回值，网络性能；

net：待仿真的神经网络；

P：网络输入；

Pi：初始输入延迟，默认为 0；

Ai：初始的层延迟，默认为 0；

T：网络目标，默认为 0。

Pi、Ai、Pf 和 Af 是可选的，它们只用于存在输入延迟和层延迟的网络。

函数中用到的信号参数采取了两种不同的形式进行表示，分别为单元阵列和矩阵的形式。其中单元阵列能够方便地对多输入多输出的神经网络进行描述。信号参数为单元阵列下的输入/输出形式为：

P：Ni×TS 维的单元阵列，每个元素 P{i,ts}都是一个 Ri×Q 的矩阵；

Pi：Ni×ID 维的单元阵列，每个元素 P{i,k} 都是一个 Ri×Q 的矩阵；

Ai：Nl×LD 维的单元阵列，每个元素 Ai{i,k}都是一个 Si×Q 的矩阵；

T：Nt×TS 维的单元阵列，每个元素 P{i,ts}都是一个 Vi×Q 的矩阵；

Y：No×TS 维的单元阵列，每个元素 Y{i,ts}都是一个 Ui×Q 的矩阵；
Pf：Ni×ID 的单元阵列，每个元素 Pf{i,k}都是一个 Ri×Q 的矩阵；
Af：Nl×LD 的单元阵列，每个元素 Af{i,k}都是一个 Si×Q 的矩阵；
E：Nt×TS 的单元阵列，每个元素 P{i,ts}都是一个 Vi×Q 的矩阵。
其中，

Ni ：神经网络输入的数目；
Nl ：神经网络层次的数目；
No ：神经网络输出的数目；
ID ：输入延迟的数目；
LD ：层次延迟的数目；
TS ：时间步长的数目；
Q ：批量；
Ri ：第 *i* 个输入的长度；
Si ：第 *i* 层的长度；
Ui ：第 *i* 个输出的长度；
Pi{i,k} ：第 *i* 个输入在 ts=k-ID 时刻的状态；
Pf{i,k} ：第 *i* 个输入在 ts=TS+k-ID 时刻的状态；
Ai{i,k} ：某层输出 *i* 在 ts=k-LD 时刻的状态；
Af{i,k} ：某层输出 *i* 在 ts=TS+k-LD 时刻的状态。

矩阵的形式只用于仿真的时间步长 TS=1 的场合，它适用于单输入单输出的网络，但也同样适用于多输入多输出的网络。在矩阵形式下，每个矩阵都是将对应的单元阵列中的元素组合而成的。其中，

P：Ri 的总和×Q 维矩阵；
Pi：Ri 的总和×(ID*Q)维矩阵；
Ai：Si 的总和×(LD*Q)维矩阵；
T：Vi 的总和×Q 维矩阵；
Y：Ui 的总和×Q 维矩阵；
Pf：Ri 的总和×(ID*Q)维矩阵；
Af：Si 的总和×(LD*Q)维矩阵；
E：Vi 的总和×Q 维矩阵。



调用格式[Y,Pf,Af] = sim(net,{Q TS},Pi,Ai)常用于一些没有输入信号的回归神经网络，如 Hopfield 网络等。

2.2.2 神经网络训练及学习函数

1）train

该函数用于对神经网络进行训练。调用格式为：

[net,tr,Y,E,Pf,Af] = train(NET,P,T,Pi,Ai)

```
[net,tr,Y,E,Pf,Af] = train(NET,P,T,Pi,Ai,VV,TV)
```

其中，

NET：待训练的神经网络；

P：网络的输入信号；

T：网络的目标，默认为 0；

Pi：初始的输入延迟，默认为 0；

Ai：初始的层次延迟，默认为 0；

VV：网络结构确认向量，默认为空；

TV：网络结构测试向量，默认为空；

net：函数返回值，训练后的神经网络；

tr：函数返回值，训练记录（包括步数和性能）；

Y：函数返回值，神经网络输出信号；

E：函数返回值，神经网络误差；

Pf：函数返回值，最终输入延迟；

Af：函数返回值，最终层延迟。

需要指出的是，参数 T 是可选的。只有当需要明确神经网络的目标时，才调用该参数。同样，Pi 和 Pf 也是可选的，它们只用于存在输入延迟和层延迟的场合。VV 和 TV 也是可选的，而且它们除了采用空矩阵之外，只能从以下范围中取值：

VV.P/ TV.P：确认/测试输入信号；

VV.T/ TV.T：确认/测试目标，默认为 0；

VV.Pi/ TV.Pi：确认/测试初始的输入延迟，默认为 0；

VV.Ai/ TV.Ai：确认/测试层确认延迟，默认为 0。



调用该函数对网络进行训练之前，需要首先设定实际的训练函数，如 trainlm 或 traingdx 等，然后该函数调用相应的算法对网络进行训练。也就是说，函数 train 只是调用设定的或者默认的训练函数对网络进行训练。

2）trainb

该函数用于神经网络权值和阈值的训练。调用格式为：

```
[NET,TR,Ac,El] = trainb(net,Pd,Tl,Ai,Q,TS,VV,TV)
info = trainb(code)
```

其中，

net：待训练的神经网络；

Pd：已延迟的输入信号；

Tl：层目标；

Ai：初始的输入；

Q：批量；

TS：时间步长；

VV：确认向量或者为空矩阵；

TV：测试向量或者为空矩阵；

NET：函数返回值，训练后的神经网络；

TR：函数返回值，每一步的训练记录，包括：

TR.epoch——仿真步次；

TR.perf——训练性能；

TR.vperf——确认性能；

TR.tperf——测试性能。

Ac：训练停止后，聚合层的输出；

El：训练停止后的层误差。

另外，训练之前需要设定以下参数，默认值见表 2-2。

表 2-2　训练参数

训练参数名称	默认值	属性
net.trainParam.epochs	100	最大训练步数
net.trainParam.goal	0	性能参数
net.trainParam.max_fail	5	确认失败的最大次数
net.trainParam.show	25	两次显示之间的的训练步数（无显示时取 NaN）
net.trainParam.time	inf	最大训练时间（单位：秒）



该函数并不能被直接调用，而是通过函数 train 隐含调用，train 通过设置网络属性 NET.trainFcn 为"trainb"来调用 trainb 对网络进行训练。

3）learnp

该函数用于神经网络权值和阈值的学习，调用格式为：

```
[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learnp(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnp(code)
```

其中，

W：S×R 维的权值矩阵（或 S×1 维的阈值向量）；

P：Q 组 R 维的输入向量（或 Q 组单个输入）；

Z：Q 组 S 维的权值输入向量；

N：Q 组 S 维的网络输入向量；

A：Q 组 S 维的输出向量；

T：Q 组 S 维的目标向量；

E：Q 组 S 维的误差向量；

gW：S×R 维的性能参数的梯度；

gA：Q 组 S 维的性能参数的输出梯度；

LP：学习参数，若没有则为空；

LS：学习状态，初始值为空；

dW：S×R 维权值（或阈值）的变化矩阵；

LS：新的学习状态；

info =learnp(code)：对不同的 code 返回相应的有用信息，包括：

pnames——返回学习参数的名称；
pdefaults——返回默认的学习参数；
needg——如果函数使用了 gW 或 gA，则返回 1。

4）learnpn

该函数也是一个权值和阈值学习函数，但它在输入向量的幅值变化非常大或者存在奇异值时，其学习速度比 learnp 要快得多。其调用格式为：

```
[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnpn(code)
```

参数含义可参见 learnp。

5）adapt

该函数使得神经网络能够自适应，调用格式为：

```
[net,Y,E,Pf,Af,tr] = adapt(NET,P,T,Pi,Ai)
```

其中，

NET：未自适应的神经网络；
P：网络输入；
T：网络目标，默认为 0；
Pi：初始输入延迟，默认为 0；
Ai：初始层延迟，默认为 0。

通过设定自适应的参数 net.adaptParam 和自适应的函数 net.adaptFunc 可调用该函数，并返回如下参数：

net：自适应后的神经网络；
Y：网络输出；
E：网络误差；
Pf：最终输入延迟；
Af：最终层延迟；
tr：训练记录（步数和性能）。



参数 T 是可选的，并且只用于必须指明网络目标的场合。同样，Pi 和 Pf 也是可选的，它们只用于存在输入延迟和层延迟的网络。

6）revert

该函数用于将更新后的权值和阈值恢复到最后一次初始化的值，调用格式为：

```
net = revert(net)
```

如果网络结构已经发生了变化，也就是说，如果网络的权值和阈值之间的连接关系，以及输入、每层的长度都与原来的网络结构有所不同，那么该函数无法将权值和阈值恢复到原来的值。在这种情况下，函数将权值和阈值都设置为 0。

2.2.3　神经网络初始化函数

1）init

该函数用于对神经网络进行初始化。调用格式为：

```
NET = init(net)
```

其中，

NET：返回参数，表示已经初始化后的神经网络；

net：待初始化的神经网络。

NET 为 net 经过一定的初始化修正而成。修正后，前者的权值和阈值都发生了改变。

2）initlay

该函数特别适用于层一层结构的神经网络的初始化。调用格式为：

```
NET = initlay(net)
info = initlay(code)
```

其中，

net：待初始化的神经网络；

NET：初始化后的神经网络；

info = initlay(code)：根据不同的 code 代码返回不同的信息，包括

pnames——初始化参数的名称；

pdefaults——默认的初始化参数。

通过指定神经网络每一层 i 的初始化函数 NET.layers{i}来调用该函数，初始化后的神经网络每一层都得到了修正。

3）initnw

该函数是一个层初始化函数，它按照 Nguyen-Widrow 准则对某层的权值和阈值进行初始化。调用格式为：

```
NET = initnw(net,i)
```

其中，

net：待初始化的神经网络；

i：层次索引；

NET：初始化后的神经网络。

4）initwb

该函数也是一个层初始化函数，它按照设定的每层的初始化函数对每层的权值和阈值进行初始化，调用格式为：

```
NET = initwb(net,i)
```

其中，

net：待初始化的神经网络；

i：层次索引；

NET：初始化后的神经网络。

例 2.1 根据给定的输入向量 P 和目标向量 T，创建一个感知器网络，对其进行训练并初始化。其中 P=[0 1 0 1; 0 0 1 1]，T=[0 0 0 1]。

MATLAB 代码如下：

```
%创建一个感知器网络，参见 2.3 节
net = newp([0 1;-2 2],1);
%感知器的 P 和阈值
net.iw{1,1}
```

```
net.b{1}
P = [0 1 0 1; 0 0 1 1];
T = [0 0 0 1];
%对感知器进行训练
net = train(net,P,T);
net.iw{1,1}
net.b{1}
%初始化感知器
net = init(net);
net.iw{1,1}
net.b{1}
```

运行结果为：

```
ans =
     0     0
ans =
     0
TRAINC, Epoch 0/100
TRAINC, Epoch 6/100
TRAINC, Performance goal met.
ans =
     1     2
ans =
    -3
ans =
     0     0
ans =
     0
```

可见，在感知器刚刚创建，尚未进行训练以前，权值和阈值都为 0；训练以后，权值和阈值分别为[1 2]和-3；对训练好的网络进行初始化后，恢复到以前的值，都为 0。

2.2.4 神经网络输入函数

1）netsum

该函数是一个输入求和函数，它通过将某一层的加权输入和阈值相加作为该层的输入。调用格式为：

```
N = netsum (Z1, Z2,…)
df = netsum('deriv')
```

其中，

Zi（i=1,2,3,…）：第 i 个输入，它的数目可以是任意个；

df = netsum('deriv')：返回的是 netsum 的微分函数 dnetsum。

2）netprod

与 netsum 的计算框架类似，不过该函数是输入求积函数，它将某一层的权值和阈值相乘作为该层的输入。调用格式为：

```
N = netprod(Z1,Z2,…)
```

```
df = netprod('deriv')
```

其中，

Zi（i=1,2,3,…）：第 i 个输入，它的数目可以是任意个；

df = netprod('deriv')：返回的是 netsum 的微分函数 dnetsum。

3）concura

该函数的作用在于使得本来不一致的权值向量和阈值向量的结构一致，以便于进行相加或相乘运算。调用格式为：

```
concur(b,q)
```

其中，

b：Nl×1 维的权值向量；

q：要达到一致化所需要的长度。

返回值为一个已经一致化了的矩阵。

例 2.2 有两个加权输入向量，Z1 和 Z2，试调用函数 netsum 将两者相加，调用 netprod 将两者相乘。

MATLAB 代码如下：

```
z1 = [1 2 4;3 4 1];
z2 = [-1 2 2; -5 -6 1];
b=[0;-1];
con1=concur(b,3);
n1 = netsum(z1,z2)
n2 = netprod(z1,z2)
n3 = netsum(z1,z2,con1)
```

运行结果为：

```
con1=[0 0 0;-1 -1 -1];
n1=[0 4 6;-2 -2 2];
n2=[-1,4,8;-15,-24,1];
n3=[0 4 6;-3 -3 1]
```

由此也可以看出 netsum 和 netprod 的运算规则，就是将权值和阈值向量中对应的元素相加或相乘。同时也显示了函数 concur 的运行机理，即修正后的矩阵就是由向量的副本组合而成的。

2.2.5 神经网络传递函数

传递函数的作用是将神经网络的输入转换为输出。

1）hardlim

该函数为硬限幅传递函数。调用格式为：

```
A = hardlim(N)
info = hardlim(code)
```

其中，

N：某层的 Q 组 S 维的输入向量；

A：返回该层的输出向量。当 N>0 时，返回值为 1；当 N<0 时，返回值为 0。

info = hardlim(code)：根据不同的代码 code 返回不同的信息，包括：

deriv——导数函数名称；

name——传递函数的全称；

output——传递函数的输出范围；

active——传递函数的输入范围。

该函数的原型函数为：

$$\text{hardlim}(n)=\begin{cases}1 & n\geqslant 0\\ 0 & n<0\end{cases}$$



可通过将 NET.layers{i}.transferFcn 设定为'hardlim'来调用该函数，这设置了神经网络中第 i 层的传递函数。

2）hardlims

该函数为对称的硬限幅传递函数。调用格式为：

```
A = hardlims(N)
info = hardlims(code)
```

其中，

N：某层的 Q 组 S 维的输入向量；

A：函数返回值。当 N>0 时，返回值为 1；当 N<0 时，返回值为 - 1。

info = hardlims(code)的含义请参见 hardlim。

该函数的原型函数为：

$$\text{hardlims}(n)=\begin{cases}1 & n\geqslant 0\\ -1 & n<0\end{cases}$$

由此可见，以上两个函数可以实现神经网络的分类和判断功能。

例 2.3 利用 MATLAB 给出一个数组，调用函数 hardlim 与 hardlims 对其进行分类。

MATLAB 代码如下：

```
n = -5:0.1:5;    %以 0.1 为步长，建立一个数组
a = hardlim(n);
b = hardlims(n);
plot(n,a,'bo');
hold on
plot(n,b,'r.');
hold on
```

代码中的函数 plot，读者可参照 MATLAB 7 的帮助进行学习，在此不再赘述。运行结果如图 2-1 所示。

图中实点部分是函数 hardlims 的分类结果；圆圈部分是 hardlim 的分类结果。可以看出，两个函数都成功地对数组进行了分类。

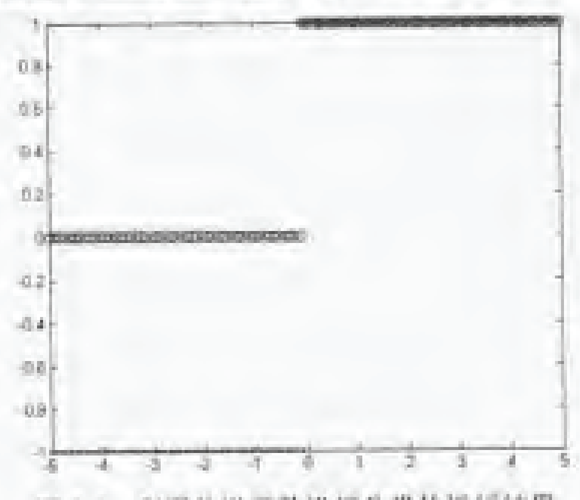


图 2-1 利用传递函数进行分类的运行结果

2.2.6 其他重要函数

1）dotprod

该函数用于对权值求点积，它求得权值与输入之间的点积作为加权输入。调用格式为：

```
Z = dotprod(W,P)
df = dotprod('deriv')
```

其中，

W：S×R 维的权值矩阵；

P：Q 组 R 维的输入向量；

Z：Q 组 R 维的 W 与 P 的点积；

df = dotprod('deriv')：返回函数的导数。

例 2.4 建立两个矩阵（或向量），演示函数 dotprod 的功能。

MATLAB 代码如下：

```
W = rand(4,3)  %建立一个 4×3 的矩阵，所有元素都小于 1
P = rand(3,1)  %建立一个 3×1 的矩阵，所有元素都小于 1
Z = dotprod(W,P)
```

运行结果为：

```
W = 0.9501    0.8913    0.8214
    0.2311    0.7621    0.4447
    0.6068    0.4565    0.6154
    0.4860    0.0185    0.7919
P = 0.9218;   0.7382;   0.1763
Z = 1.6786;   0.8540;   1.0048;   0.6012
```

2.3 感知器的神经网络工具箱函数

感知器网络是由美国计算机科学家罗森布拉特于 1957 年提出的。从某种意义上讲，感知器可以说是最早的人工神经网络。单层感知器是一个具有单层神经元、采用阈值激活函

数的前向网络。通过对网络权值的训练，可以使感知器对一组输入矢量的响应达到元素为 0 或 1 的目标输出，从而实现对输入矢量进行分类的目的。

MATLAB 7 的神经网络工具箱中提供了大量的感知器函数，下面我们将对这些函数的功能、调用格式、使用方法及注意事项作详细说明。

常用的感知器函数见表 2-3。

表 2-3 感知器常用函数

函数类别	函数名称	函数用途
感知器创建函数	newp	创建一个感知器网络
显示函数	plotpc	在感知器向量图中绘制分界线
	plotpv	绘制感知器的输入向量和目标向量
性能函数	mae	平均绝对误差函数

2.3.1 感知器创建函数

可通过感知器生成函数来创建一个感知器，并且可对感知器进行初始化、仿真和训练等。

感知器生成函数 newp 用于创建一个感知器网络，调用格式为：

```
net = newp
net = newp(pr,s,tf,lf)
```

其中，

net：函数返回参数，表示生成的感知器网络；

net=newp：表示在一个对话框中定义感知器的属性；

pr：一个 R×2 的矩阵，由 R 组输入向量的最大值和最小值组成；

s：神经元的个数；

tf：感知器的传递函数，可选参数为 hardlim 和 hardlims，默认为 hardlim；

lf：感知器的学习函数，可选参数为 learnp 和 learnpn，默认情况下为 learnp。

2.3.2 显示函数

1）分界线绘制函数 plotpc

该函数用于在感知器向量图中绘制分界线，其调用格式为：

```
plotpc(W,b)
plotpc(W,b,h)
```

其中，

W：S×R 维的加权矩阵（R 必须小于等于 3）；

b：S×1 维的阈值向量；

h：最后画线的控制权；

plotpc(W,b)：返回的是对所绘制分界线的控制权；

plotpc(W,b,h)：用于在绘制新线之前检查最新绘制的分界线。


该函数一般在 plotpv 函数之后调用，而且并不改变现有的坐标轴标准。

2）输入/目标向量绘制函数 plotpv

该函数用于绘制感知器的输入向量和目标向量，调用格式为：

```
plotpv(p,t)
plotpv(p,t,v)
```

其中，

p：Q 组 R 维的输入向量；

t：Q 组 S 维的双目标向量；

v=[x_min x_max y_min y_max]：图形的最大值，绘制工作必须位于 v 所限定的范围中；

plotpv(p,t)：以 t 为标尺，绘制 p 的列向量；

plotpv(p,t,v)：在 v 的范围中绘制 p 的列向量。

例 2.5 本例的主要目的在于演示函数 plotpc 和 plotpv 的应用。

假定已给出了某感知器的输入变量 p 和目标变量 t，绘制其曲线；然后给定感知器的权值和阈值，绘制其分界线。

MATLAB 代码如下：

```
p = [0 0 1 1; 0 1 0 1];
t = [0 0 0 1];
plotpv(p,t) %绘制输入向量和目标向量
net = newp(minmax(p),1); %创建一个感知器网络
net.iw{1,1} = [-1.2 -0.5]; %设定权值
net.b{1} = 1; %设定阈值
plotpc(net.iw{1,1},net.b{1})
```

其中，minmax 为数学函数。本例中 minmax(p)=[0 1;0 1]。

运行结果如图 2-2 所示。

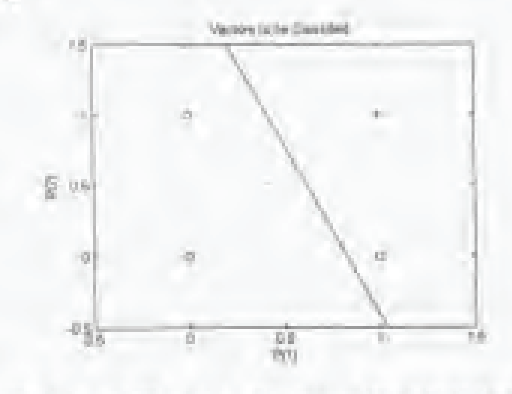


图 2-2 函数 plotpc 和 plotpv 演示运行结果

2.3.3 性能函数

1）性能函数 mae

该函数以平均绝对误差为准则，确定神经网络性能的函数。调用格式为：

```
perf = mae(e,x,pp)
perf = mae(e,net,pp)
info = mae(code)
```

其中，

e：误差向量矩阵（或向量）；

x：所有的权值和阈值向量，可忽略；

pp：性能参数，可忽略；

net：待评定的神经网络；

perf：函数的返回值，为平均绝对误差；

mae(code)：根据 code 值的不同，返回不同的信息，包括：

deriv——返回导数函数的名称；

name——返回函数全称；

pnames——返回训练参数的名称；

pdefaults——返回默认的训练参数。

例 2.6 感知器最重要的也是最实用的功能是对输入向量进行分类。本例尝试建立一个感知器模型，实现电路中“或”门的功能，从而实现对输入的分类。

“或”门输入/输出见表 2-4，由此可得出网络的输入向量 P 和目标向量 T。其中，P = [0 0 1 1; 0 1 0 1]; T = [0 1 1 1]。

表 2-4　“或”门输入/输出

输　入	输　出
0 0	0
0 1	1
1 0	1
1 1	1

本例的 MATLAB 代码为：

```
P = [0 0 1 1; 0 1 0 1];
T = [0 1 1 1];
net=newp(minmax(P),1);
Y = sim(net,P)
net.trainParam.epochs = 20;
net = train(net,P,T);
Y = sim(net,P)
err1=mae(Y-T)
```

输出为：

```
Y =
     1     1     1     1
TRAINC, Epoch 0/20
TRAINC, Epoch 4/20
TRAINC, Performance goal met.
```

```
Y =
      0     1     1     1
err1 =
      0
```

由此可见，感知器在训练以前的输出是不符合要求的，经过 4 次训练后的输出已经和目标向量一致了，训练过程的误差曲线如图 2-3 所示。

本例创建的感知器只有一个神经元，这里也符合了神经网络的设计原则，即在设计神经网络时，在同样可以完成任务的情况下，尽量采用简单的结构。因为结构简单的神经网络计算负担轻，运行速度一般比较快。感知器的传递函数和学习函数都采用默认值，分别为 hardlim 和 learnp，这因为网络的输出为 0-1 的二值结构，只有采用 hardlim 才满足要求，输入向量中不存在奇异值，元素之间的距离也比较小，因此，采用 learnp 就足够了。

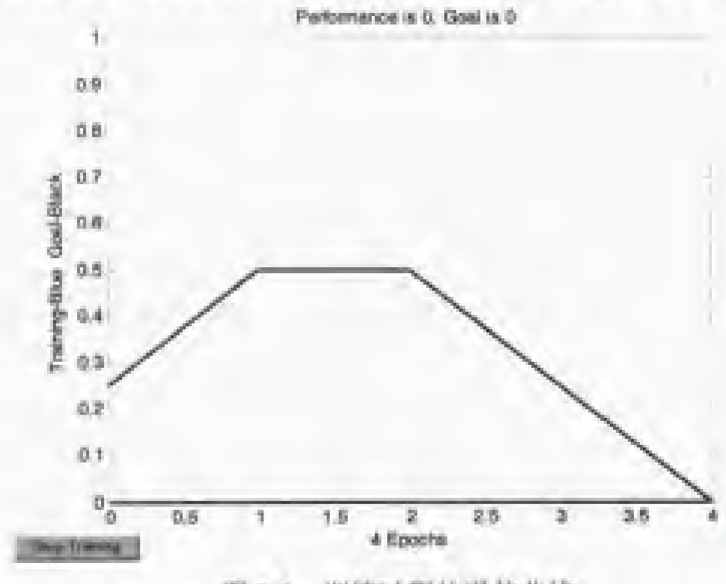


图 2-3　训练过程的误差曲线

利用 trainc 函数对网络进行训练，训练的结果非常理想，训练后的网络成功实现了“或”功能。利用平均绝对误差函数 mae 计算网络的性能，结果为 0，从另一个方面说明了网络的性能是非常好的。

利用自适应函数 adapt 同样可以达到训练的效果。采用以下的代码对网络进行训练：

```
while mae(e)
    [net,Y,e]=adapt(net,P,T);
end
Y=sim(net,P)
```

网络的输出为：

```
Y =
      0     1     1     1
```

训练过程中采用 mae 函数来得到网络误差，以此作为是否停止训练的标准。此时的网络输出和前面的一致，因此，采用这种方式对网络进行训练是很成功的。

例 2.7 本例的目的在于演示标准学习函数 learnpn 的应用。要求建立一个感知器网络，对一组存在奇异值的输入向量 P 进行分类。

网络的输入向量 P 和目标向量 T 分别为：

```
P = [0 0.5 1.2 20; 0 0.6 1.1 90];
T = [0 1 0 1];
```

其中，T 表明了输入向量的分类情况。输入的向量分布如图 2-4 所示，其中的一个坐标点与其他点的距离都很大，而另外三个点相对比较集中。因此，输入向量中存在奇异值。

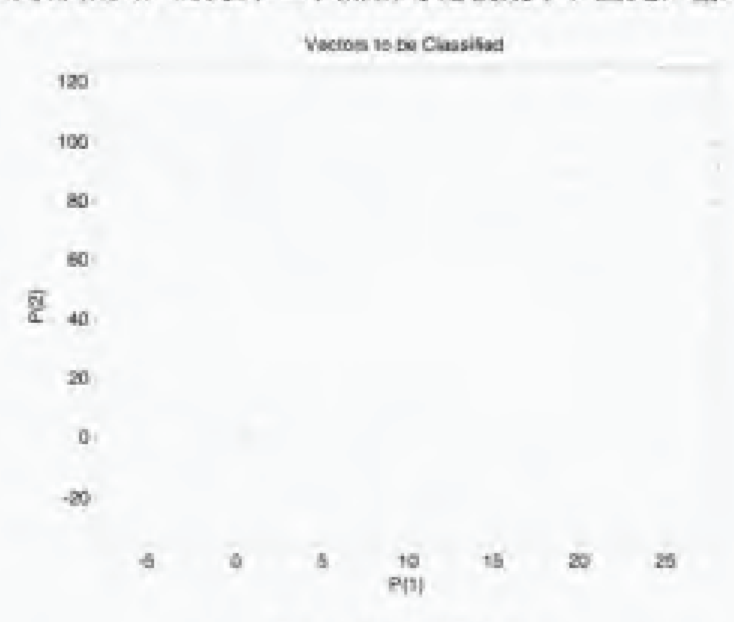


图 2-4 输入向量分布

本例的 MATLAB 代码为：

```
P = [0 0.5 1.2 20; 0 0.6 1.1 90];
T = [0 1 0 1];
net=newp(minmax(P),1);
net.trainParam.epochs=200;
net=train(net,P,T);
figure;
plotpv(P,T);
plotpc(net.iw{1},net.b{1});
```

训练过程中的误差曲线如图 2-5 所示。可见，网络的训练过程出现了振荡，而且训练次数也比较多。

训练得到的分类线如图 2-6 所示，由于奇异值的存在，样本点相对比较集中，因此无法看出网络的分类性能。接下来将局部放大，检查网络的分类性能。代码如下：

```
figure;
plotpv(P,T);
plotpc(net.iw{1},net.b{1});
%限制坐标起点和终点
axis([-2 2 -2 2])
```

局部放大后的分类线如图 2-7 所示，两类样本点中分别有一个位于分类线上，由此可见，输入样本中的奇异点对于网络的分类性能影响很大。

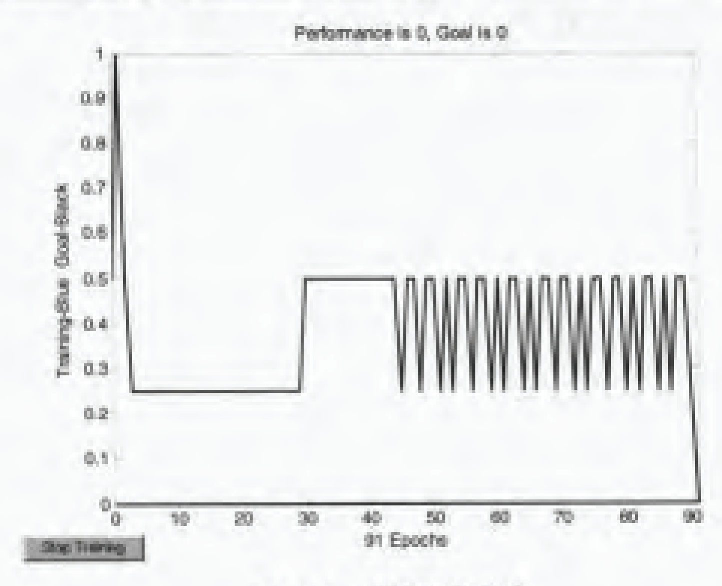


图 2-5 训练误差曲线

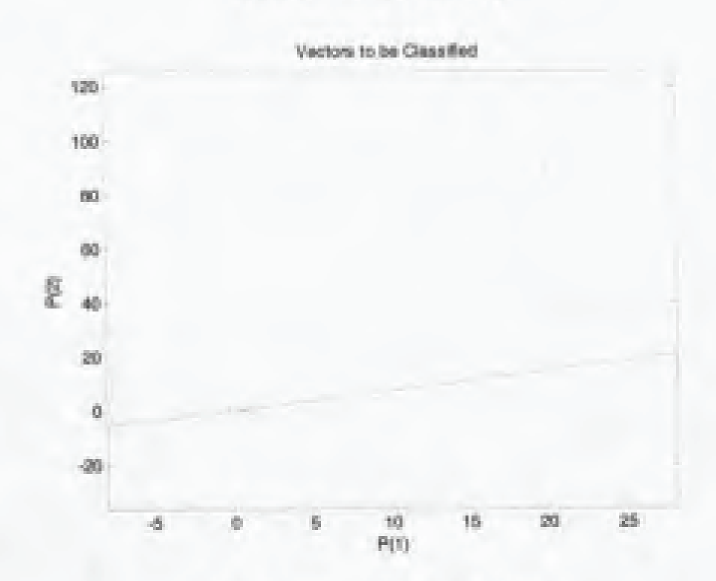


图 2-6 训练得到的分类线

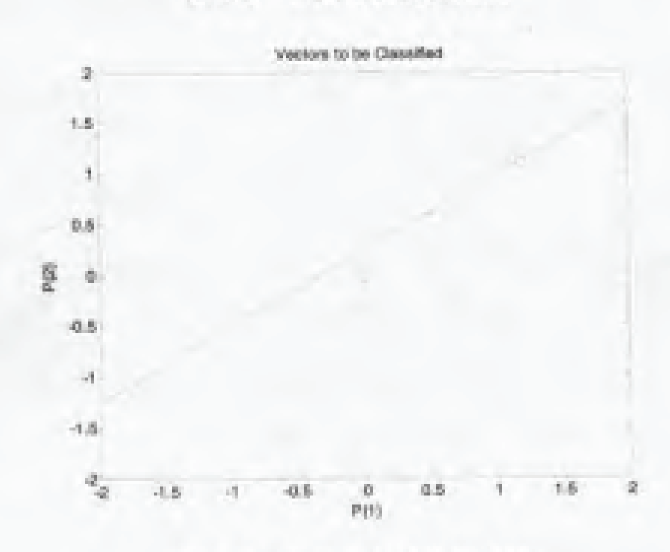


图 2-7 局部放大后的分类线

以上代码创建的感知器的学习函数为 learnp，接下来利用标准训练函数 learnpn 作为网络的学习函数。除感知器创建代码有区别外，其余的全部一致，代码为：

```
net=newp(minmax(P),1,'hardlim','learnpn');
```

网络的训练误差曲线如图 2-8 所示，由图可见，网络经过 83 次学习后性能为 0，相对于图 2-5 中的 91 次，网络的训练时间有所减少。

此时网络得到的分类线如图 2-9 所示，可见此时的分类线比图 2-7 中的分类线更加准确，两类样本点被准确地分开。

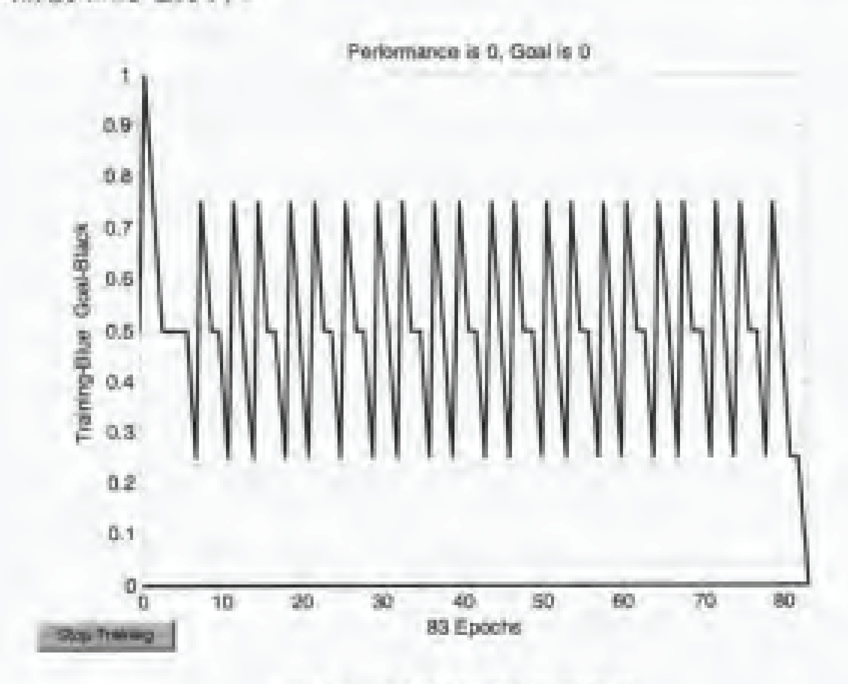


图 2-8 网络的训练误差曲线

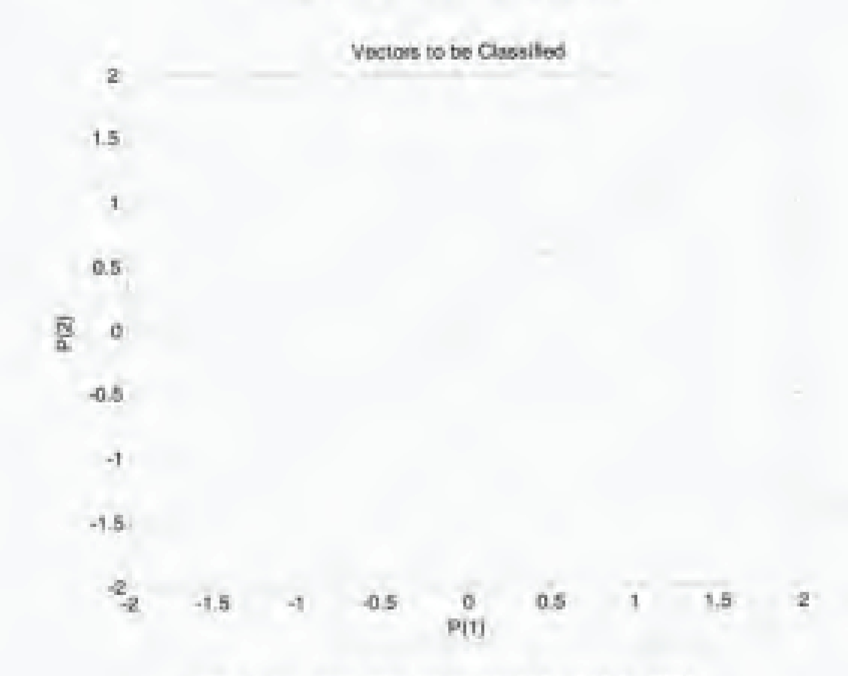


图 2-9 网络得到的局部放大后的分类线

2.4 BP 网络的神经网络工具箱函数

BP 网络的全称为 Back-Propagation Network，即反向传播网络。除非有特别注明，本书中一律使用 BP 网络作为反向传播网络的简称。

BP 网络是利用非线性可微分函数进行权值训练的多层网络。它包含了神经网络理论中最为精华的部分，由于其结构简单、可塑性强，故在函数逼近、模式识别、信息分类及数据压缩等领域得到了广泛的应用。特别的，由于它的数学意义明确，学习算法步骤分明，使得应用背景更加广泛。

MATLAB 7 神经网络工具箱中包含了许多用于 BP 网络分析与设计的函数，本节将详细说明这些函数的功能、调用格式、参数特性和注意事项等。BP 网络的常用函数如表 2-5 所示。

表 2-5　BP 网络的常用函数表

函数类型	函数名称	函数用途
前向网络创建函数	newcf	创建级联前向网络
	newff	创建前向 BP 网络
	newfftd	创建存在输入延迟的前向网络
传递函数	logsig	S 型的对数函数
	dlogsig	logsig 的导函数
	tansig	S 型的正切函数
	dtansig	tansig 的导函数
	purelin	纯线性函数
	dpurelin	Purelin 的导函数
学习函数	learngd	基于梯度下降法的学习函数
	learngdm	梯度下降动量学习函数
性能函数	mse	均方误差函数
	msereg	均方误差规范化函数
显示函数	plotperf	绘制网络的性能
	plotes	绘制一个单独神经元的误差曲面
	plotep	绘制权值和阈值在误差曲面上的位置
	errsurf	计算单个神经元的误差曲面

2.4.1 BP 网络创建函数

1）newcf

该函数用于创建级联前向 BP 网络，调用格式为：

```
net = newcf
net = newcf(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

其中，

net= newcf：用于在对话框中创建一个 BP 网络；

PR：由每组输入（共有 R 组输入）元素的最大值和最小值组成的 R×2 维的矩阵；

Si：第 i 层的长度，共计 Nl 层；

TFi：第 i 层的传递函数，默认为“tansig”；

BTF：BP 网络的训练函数，默认为“trainlm”；

BLF：权值和阈值的 BP 学习算法，默认为“learngdm”；

PF：网络的性能函数，默认为“mse”。



参数 TFi 可以采用任意的可微传递函数，比如 tansig、logsig 和 purelin 等；训练函数可以是任意的 BP 训练函数，如 trainlm、trainbfg、trainrp 和 traingd 等。值得指出的是，BTF 默认采用 trainlm 是因为该函数的速度很快，但该函数的一个重要缺陷是运行过程会消耗大量的内存资源。如果您的计算机内存不够大，不建议采用 BTF 的默认函数 trainlm，而建议采用训练函数 trainbfg 或 trainrp，虽然这两个函数的运行速度比较慢，但它们的共同特点是内存占用量小，不至于出现训练过程死机的情况。

2）newff

该函数用于创建一个 BP 网络。调用格式为：

```
net = newff
net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

其中，

net= newff：用于在对话框中创建一个 BP 网络。

其他参数含义请参见函数 newcf。

3）newfftd

该函数用于创建一个存在输入延迟的前向网络。调用格式为：

```
net = newfftd
net = newfftd(PR,ID,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

其中，

NET = NEWFFTD：用于在对话框中创建一个 BP 网络。

其他参数含义请参见 newcf。

2.4.2 神经元上的传递函数

传递函数是 BP 网络的重要组成部分。传递函数又称为激活函数，必须是连续可微的。BP 网络经常采用 S 型的对数或正切函数和线性函数。

1）logsig

该传递函数为 S 型的对数函数。调用格式为：

```
A = logsig(N)
info = logsig(code)
```

其中，

N：Q 个 S 维的输入列向量；

A：函数返回值，位于区间（0,1）中；

info = logsig(code)：依据 code 值的不同返回不同的信息，包括：

deriv——返回微分函数的名称；

name——返回函数全称；

output——返回输出值域；

active——返回有效的输入区间。

下面以一组简单的代码产生一个对数 S 型的传递函数为例：

```
n=-10:0.1:10;
a=logsig(n)
plot(n,a)
grid
```

运行结果如图 2-10 所示。可见，函数 logsig 可将神经元的输入（范围为整个实数集）映射到区间（0,1）中，又由于该函数为可微函数，因此非常适合于利用 BP 算法训练神经网络。



MATLAB 7 按照此形式的函数来计算对数传递函数的值：$n = 2/\left(1+\exp\left(-2n\right)\right)$。



图 2-10　logsig 函数运行结果

2）dlogsig

该函数为 logsig 的导函数。调用格式为：

```
dA_dN = dlogsig(N,A)
```

其中，

N：S×Q 维网络输入；

A：S×Q 维网络输出；

dA_dN：函数返回值，输出对输入的导数。



MATLAB 7 按照此形式的函数来计算对数传递函数的值：$y = x*\left(1-x\right)$。

例 2.8 现有某 BP 网络某层的 3 个神经元，其传递函数均为 S 型的对数函数，试利用

函数 dlogsig 求解输出对输入的导数。

MATLAB 代码如下：

```
N = [0.1; 0.8; -0.7];
A = logsig(N)
dA_dN = dlogsig(N,A)
```

运行结果如下：

```
A =
    0.5250
    0.6900
    0.3318
dA_dN =
    0.2494
    0.2139
    0.2217
```

3）tansig

该函数为双曲正切 S 型传递函数。调用格式为：

```
A = tansig(N)
info = tansig(code)
```

其中，

N：Q 个 S 维的输入列向量；

A：函数返回值，位于区间（-1,1）之间。

info = tansig(code)的含义请参见 info = logsig(code)。

以下代码可绘制一个双曲正切 S 型传递函数：

```
n = -10:0.1:10;
a = tansig(n);
plot(n,a)
grid
```

运行结果如图 2-11 所示。

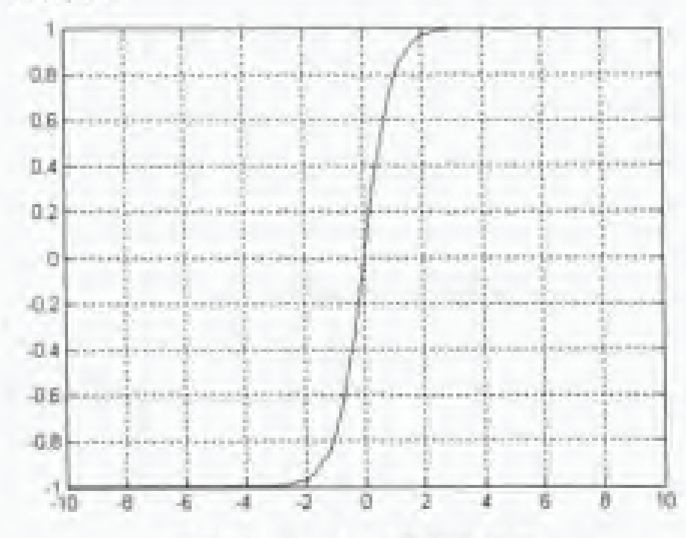


图 2-11 tangsig 函数运行结果



MATLAB 7 按照此形式的函数来计算该双曲正切传递函数的值：$n = 2/(1+\exp(-2n))-1$。

4）dtansig

该函数为 tansig 的导函数。调用格式为：

```
dA_dN = dtansig(N,A)
```

各参数的意义请参见 dlogsig，其原型函数为 $y = 1 - x^2$。

5）purelin

该函数为线性传递函数。调用格式为：

```
A = purelin(N)
info = purelin(code)
```

其中，

N：Q 个 S 维的输入列向量；

A：函数返回值，A=N。

info = purelin(code)的含义请参见 info = logsig(code)。

下面以一组简单的代码产生一个线性 S 型传递函数为例：

```
n = -10:0.1:10;
a = purelin(n);
plot(n,a)
grid
```

运行结果如图 2-12 所示。

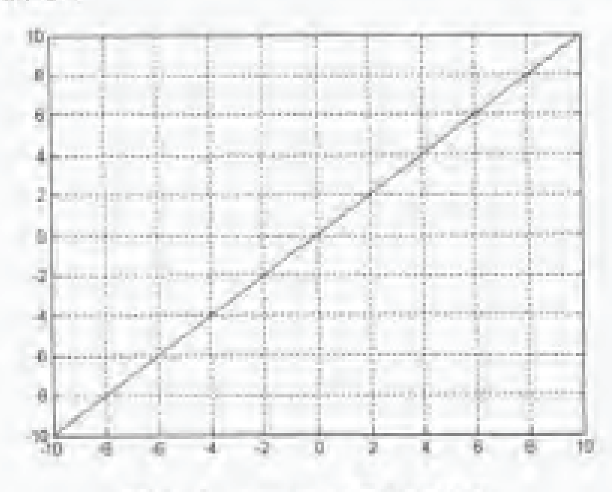


图 2-12　purelin 函数运行结果



MATLAB 7 按照此形式的函数来计算函数 purelin：$y = x$。

6）dpurelin

该函数为 purelin 的导函数。调用格式为：

```
dA_dN = dpurelin(N,A)
```

各参数的含义请参见 dlogsig，原型函数为常数函数 $x = 1$。

2.4.3 BP网络学习函数

1）learngd

该函数为梯度下降权值/阈值学习函数，它通过神经元的输入和误差，以及权值和阈值的学习速率，来计算权值或阈值的变化率。调用格式为：

```
[dW,ls] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,ls] = learngd(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngd(code)
```

其中，

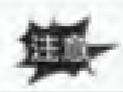
W：S×R维的权值矩阵；
b：S维的阈值向量；
P：Q组R维的输入向量；
ones(1,Q)：产生一个Q维的输入向量；
Z：Q组S维的加权输入向量；
N：Q组S维的输入向量；
A：Q组S维的输出向量；
T：Q组S维的层目标向量；
E：Q组S维的层误差向量；
gW：与性能相关的S×R维梯度；
gA：与性能相关的S×R维输出梯度；
D：S×S维的神经元距离矩阵；
LP：学习参数，可通过该参数设置学习速率，设置格式如LP.lr=0.01；
LS：学习状态，初始状态下为空；
dW：S×R维的权值或阈值变化率矩阵；
db：S维的阈值变化率向量；
ls：新的学习状态；
learngd(code)：根据不同的code值返回有关函数的不同信息，包括：
pnames——返回设置的学习参数；
pdefaults——返回默认的学习参数；
needg——如果函数使用了gW或gA，则返回1。

2）learngdm

该函数为梯度下降动量学习函数，它利用神经元的输入和误差、权值或阈值的学习速率和动量常数，来计算权值或阈值的变化率。调用格式为：

```
[dW,LS] = learngdm(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learngdm(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngdm(code)
```

各参数的含义请参见learngd。



动量常数mc是通过学习参数LP设置的，格式为lp.mc = 0.8。

2.4.4 BP 网络训练函数

1）trainbfg

该函数为 BFGS 准牛顿 BP 算法函数。除了 BP 网络外，该函数也可以训练任意形式的神经网络，只要它的传递函数对于权值和输入存在导函数即可。调用格式为：

```
[net,TR,Ac,El] = trainbfg(NET,Pd,Tl,Ai,Q,TS,VV,TV)
info = trainbfg(code)
```

其中，

NET：待训练的神经网络；

Pd：有延迟的输入向量；

Tl：层次目标向量；

Ai：初始的输入延迟条件；

Q：批量；

TS：时间步长；

VV：确认向量结构或者为空；

TV：检验向量结构或者为空；

net：训练后的神经网络；

TR：每步训练的有关信息记录，包括：

TR.epoch——时刻点；

TR.perf——训练性能；

TR.vperf——确认性能；

TR.tperf——检验性能。

Ac：上一步训练中聚合层的输出；

El：上一步训练中的层次误差；

info = trainbfg(code)：根据不同的 code 值返回不同的有关 trainbfg 的信息，包括：

pnames——返回设定的训练参数；

pdefaults——返回默认的训练参数。

在利用该函数进行 BP 网络训练时，MATLAB 7 已经默认了以下训练参数，如表 2-6 所示。

表 2-6 BP 网络训练参数

参数名称	默认值	属性
net.trainParam.epochs	100	训练次数，100 为训练次数的最大值，人工设定的训练次数不能超过 100
net.trainParam.show	25	两次显示之间的训练步数（无显示时设为 NaN）
net.trainParam.goal	0	训练目标
net.trainParam.time	inf	训练时间，inf 表示训练时间不限
net.trainParam.min_grad	1e-6	最小性能梯度
net.trainParam.max_fail	5	最大确认失败次数
net.trainParam.searchFcn	'srchcha'	所用的线性搜索路径

2）traingd

该函数为梯度下降 BP 算法函数。调用格式为：

```
[net,tr,Ac,El] = traingd(net,Pd,Tl,Ai,Q,TS,VV,TV)
info = traingd(code)
```

其参数意义、设置格式和适用范围等请参见 trainbfg。

3）traingdm

该函数为梯度下降动量 BP 算法函数。调用格式为：

```
[net,tr,Ac,El] = traingdm(net,Pd,Tl,Ai,Q,TS,VV,TV)
info = traingdm(code)
```

其参数意义、设置格式和适用范围等请参见 trainbfg。

此外，MATLAB 7 的神经网络工具箱中还有一系列训练函数可用于对 BP 网络的训练，限于篇幅的原因，我们在此以表格的形式将它们表示出来，见表 2-7。读者可以比照函数 trainbfg 的调用格式进行学习使用。

表 2-7　其他训练函数

函 数 名 称	函 数 说 明
trainbr	Bayes 规范化 BP 训练函数
trainc	循环顺序渐增训练函数
traincgb	Powell-Beale 连接梯度 BP 训练函数
traincgf	Fletcher-Powell 连接梯度 BP 训练函数
traincgp	Polak-Ribiere 连接梯度 BP 训练函数
traingda	自适应 lrBP 的梯度递减训练函数
traingdx	动量及自适应 lrBP 的梯度递减训练函数
trainlm	Levenberg-Marquardt BP 训练函数
trainoss	一步正切 BP 训练函数
trainr	随机顺序递增更新训练函数
trainrp	带反弹的 BP 训练函数
trains	顺序递增 BP 训练函数
trainscg	量化连接梯度 BP 训练函数



以上这些训练函数不仅可用于 BP 网络的训练，还适用于其他任何神经网络，只要其传递函数对于权值和阈值存在导函数即可。

2.4.5　性能函数

1）mse

该函数为均方误差性能函数，调用格式为：

```
perf = mse(e,x,pp)
perf = mse(e,net,pp)
```

```
info = mse(code)
```

各参数含义请参见 mae。

2）msereg

该函数也是性能函数，它通过两个因子的加权和来评价网络的性能，这两个因子分别是均方误差、均方权值和阈值。调用格式为：

```
perf = msereg(e,x,pp)
perf = msereg(e,net)
info = msereg(code)
```

各参数含义请参见 mae。



在使用该函数前，需要设定性能参数 pp，格式为 PP.ratio=0.3，该参数的意义是误差相对于权值和阈值的重要性。这样一来，函数的返回值=均方误差 × PP.ratio + 均方权值和阈值 × PP.ratio。

例 2.9 创建一个 BP 网络，并评估其性能。

MATLAB 代码如下：

```
%创建一个 BP 网络
net = newff([-2 2],[4 1],{'tansig','purelin'},'trainlm','learngdm','msereg');
p = [-2 -1 0 1 2];
t = [0 1 1 1 0];
y = sim(net,p)
e = t-y   %误差向量
net.performParam.ratio = 20/(20+1);   %设置性能参数
perf = msereg(e,net)
```

运行结果为：

```
y =
   0.9027     0.2892    - 0.4402     1.1509     0.7574
e =
  - 0.9027     0.7108     1.4402    -0.1509    - 0.7574
perf =
   1.1389
```

2.4.6 显示函数

1）plotperf

该函数用于绘制网络的性能。调用格式为：

```
plotperf(tr,goal,name,epoch);
```

其中，

tr：网络训练记录；

goal：性能目标，默认为 NaN；

name：训练函数名称，默认为空；

epoch：训练步数，默认为训练记录的长度。

函数除了可以绘制网络的训练性能外，还可以绘制性能目标、确认性能和检验性能。

当然，前提条件是它们都存在。

这里有一个输入向量 P 和目标向量 T，分别有 8 个向量。由此导出一组确认样本，分别为：

```
P = 1:8; T = sin(P);
VV.P = P; VV.T = T+rand(1,8)*0.1;
```

创建一个 BP 网络，并进行训练，找出 P 和 T 之间的非线性关系，并利用确认样本对网络进行检验。

```
net = newff(minmax(P),[4 1],{'tansig','tansig'});
[net,tr] = train(net,P,T,[],[],VV);
```

网络的训练误差曲线如图 2-13 所示，训练结果为：

```
TRAINLM, Epoch 0/100, MSE 0.951747/0, Gradient 3.00875/1e-010
TRAINLM, Epoch 18/100, MSE 0.0652647/0, Gradient 0.00312702/1e-010
TRAINLM, Validation stop.
```

在训练过程中，直接调用 plotperf 函数显示网络的训练记录。当然，也可以直接调用 plotperf 绘制网络的训练记录，显示结果与图 2-13 是一致的。

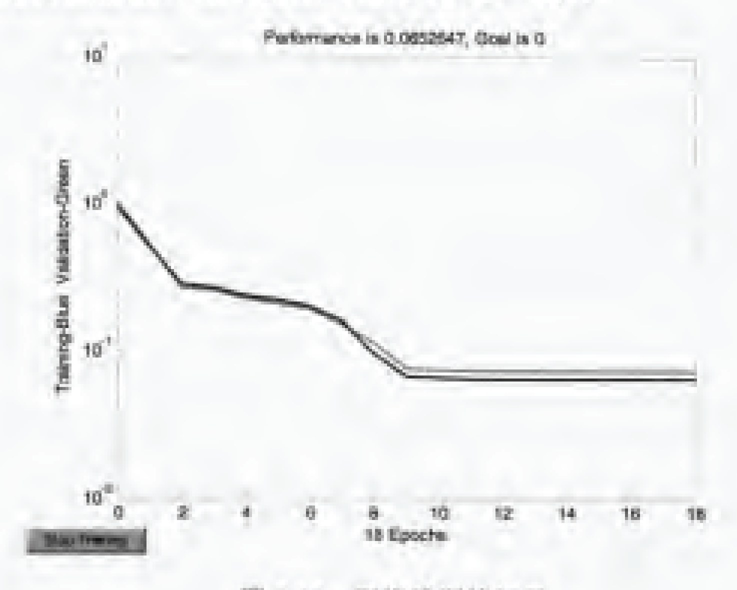


图 2-13　网络的训练记录

2）plotes

该函数用于绘制一个单独神经元的误差曲面。调用格式为：

```
plotes(wv,bv,es,v)
```

其中，

wv：权值的 N 维行向量；

bv：M 维的阈值行向量；

es：误差向量组成的 M×N 维矩阵；

v：视角，默认为[-37.5, 30]。

函数绘制的误差曲面图是由权值和阈值确定的、由函数 errsurf 计算得出的。

3）plotep

该函数用于绘制权值和阈值在误差曲面上的位置。调用格式为：

```
H = plotep(w,b,e)
```

```
H = plotep(w,b,e,h)
```

其中，

w：当前权值；

b：当前阈值；

e：当前单输入神经元的误差；

h：权值和阈值在上一时刻的位置信息向量；

H：当前的权值和阈值位置信息向量。



在整个绘制过程中，H可以看做一个“临时”存储变量。在绘制新的权值和阈值位置之前，H被清除；在绘制后，H记录了本次权值和阈值的位置信息。

4）errsurf

此函数用于计算单个神经元的误差曲面。调用格式为：

```
E = errsurf (P,T,WV,BV,F)
```

其中，

P：输入行向量；

T：目标行向量；

WV：权值列向量；

BV：阈值列向量；

F：传递函数的名称。

神经元的误差曲面是由权值和阈值的行向量确定的。

下面的一组代码可以分析一个BP网络中某个神经元的误差，并绘制出其误差曲面与轮廓线。

```
p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
wv = -1:.1:1;
bv = -2.5:.25:2.5;
es = errsurf(p,t,wv,bv,'logsig');
plotes(wv,bv,es,[60 30])
```

运行结果如图2-14所示。

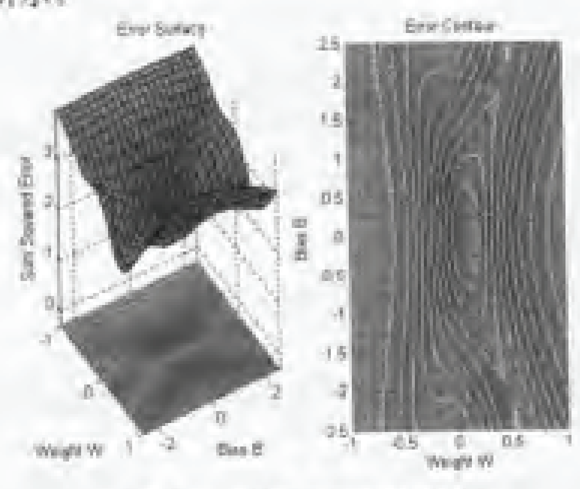


图2-14 误差曲面和轮廓线

例 2.10 BP 网络的一个重要功能就是非线性映射的能力，这一功能非常适合于函数逼近等，也就是说，找出两组数据之间的关系。本例给出输入向量 P 和目标向量 T，建立一个 BP 网络，找出 P 和 T 之间的关系。

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4]
```

利用 newff 创建一个 BP 网络：

```
net = newff([0 10],[5 1],{'tansig' 'purelin'});
```

可见，网络的中间层有 5 个神经元，传递函数为 tansig；输出层有 1 个神经元，传递函数为 purelin。BP 网络中间层神经元的数目对网络性能有着比较大的影响，需要通过不断地尝试才能确定。由于没有特别设定训练函数，因此训练函数取默认值 trainlm。

图 2-15 中的“o”表示对训练前的网络进行仿真得到的输出，直线表示输入向量和目标向量之间的函数关系。由此可见，在训练以前，网络的非线性映射性能是很差的。



图 2-15　训练前网络的输出

接下来对网络进行训练，代码如下：

```
net.trainParam.epochs = 200;
net = train(net,P,T);
figures;
Y = sim(net,P);
plot(P,T,P,Y,'o')
```

网络的训练结果为：

```
TRAINLM, Epoch 0/200, MSE 9.65663/0, Gradient 60.1766/1e-010
TRAINLM, Epoch 25/200, MSE 0.00908584/0, Gradient 0.0482997/1e-010
TRAINLM, Epoch 50/200, MSE 0.00401579/0, Gradient 0.129008/1e-010
TRAINLM, Epoch 75/200, MSE 0.00186609/0, Gradient 0.0186651/1e-010
TRAINLM, Epoch 100/200, MSE 0.000907903/0, Gradient 0.031521/1e-010
TRAINLM, Epoch 125/200, MSE 0.000103122/0, Gradient 0.459816/1e-010
TRAINLM, Epoch 150/200, MSE 3.26443e-005/0, Gradient 0.00822781/1e-010
TRAINLM, Epoch 175/200, MSE 1.50462e-005/0, Gradient 0.0823667/1e-010
TRAINLM, Epoch 200/200, , Gradient 0.00641989/1e-010
```

TRAINLM, Maximum epoch reached, performance goal was not met.

由此可见，经过 200 次训练后，虽然网络的性能还没有为 0，但是输出的均方误差已经很小了，MSE=5.04746e-006/0，误差曲线如图 2-16 所示。因此，此时的网络输出应该是比较精确的，如图 2-17 所示。

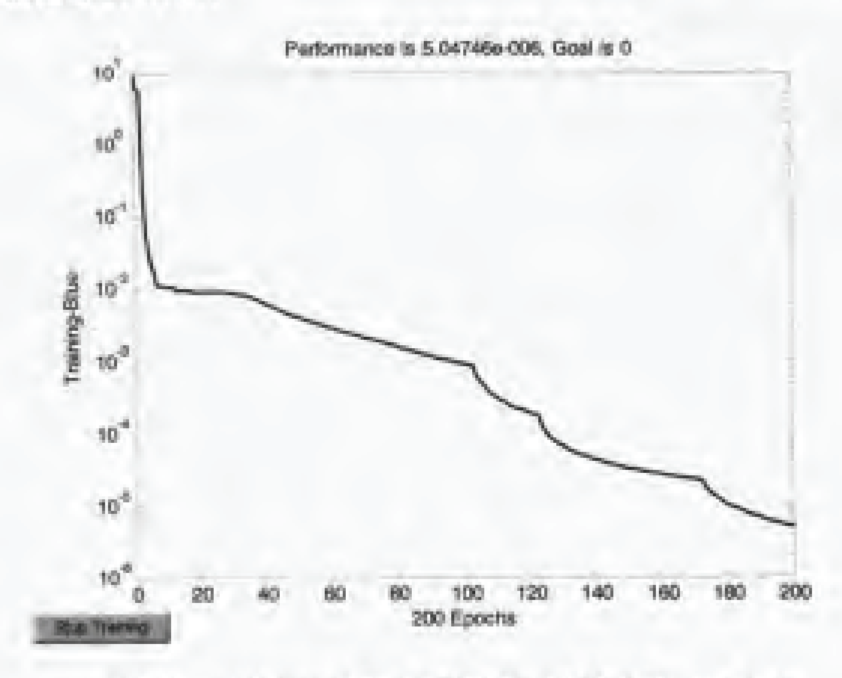


图 2-16 训练得到的均方误差曲线（训练函数：trainlm）

图 2-17 显示的结果证实了我们的猜测，此时网络对 P 和 T 之间非线性映射关系的拟合是非常精确的。

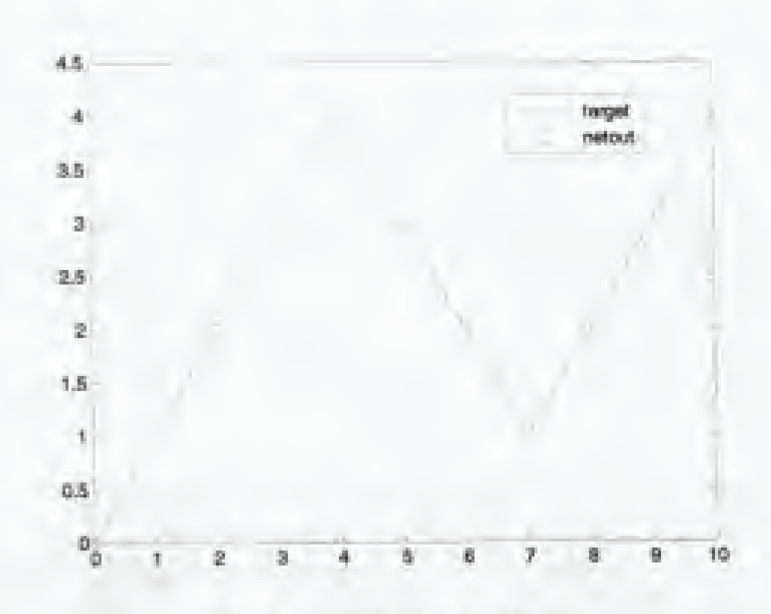


图 2-17 训练后的网络输出

上面创建的 BP 网络的学习函数、训练函数和性能函数都采用默认值，分别为 learngdm、trianlm 和 mse。接下来，还是针对这个问题，建立一个学习函数为 learngd、训练函数为 traingd 和性能函数为 msereg 的 BP 网络。

```
net = newff([0 10],[5 1],{'tansig' 'purelin'},'traingd','learngd','msereg');
```

此时网络的训练误差曲线如图 2-18 所示，可见，经过 200 次训练后，网络的输出误差比较大，而且网络误差的收敛速度非常慢。这是由于训练函数 traingd 为单纯的梯度下降训练函数，训练速度比较慢，而且容易陷入局部最小的情况。此时网络对 P 和 T 之间关系的拟合情况输出如图 2-19 所示。

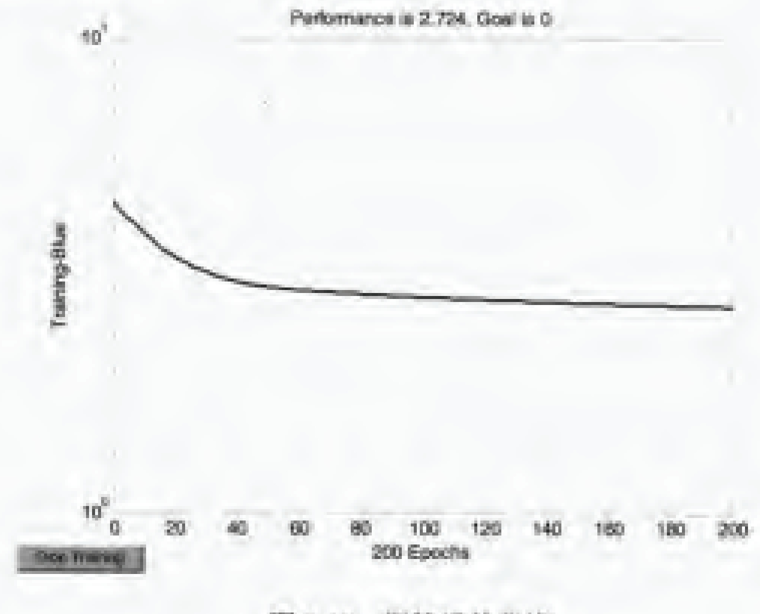


图 2-18　训练误差曲线

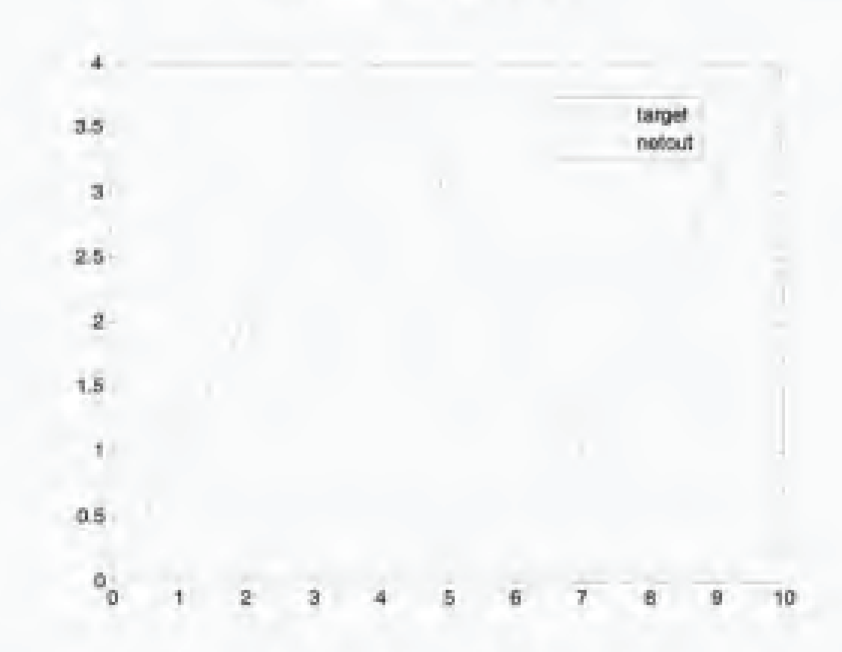


图 2-19　网络的输出（训练函数：traingd）

可见网络的精度确实比较差。这种情况下，将训练函数修改为 traingdx，该函数也是梯度下降法训练函数，但是在训练过程中，它的学习速率是可变的。

```
net = newff([0 10],[5 1],{'tansig' 'purelin'},'traingdx','learngd','msereg');
```

此时，网络的训练误差曲线如图 2-20 所示，在 200 次训练后，以 msereg 函数评价的

网络性能为 0.877799，这已经不是很大了。可见，采用学习速率可变的训练函数对网络进行训练，训练后网络的性能应该是不错的。

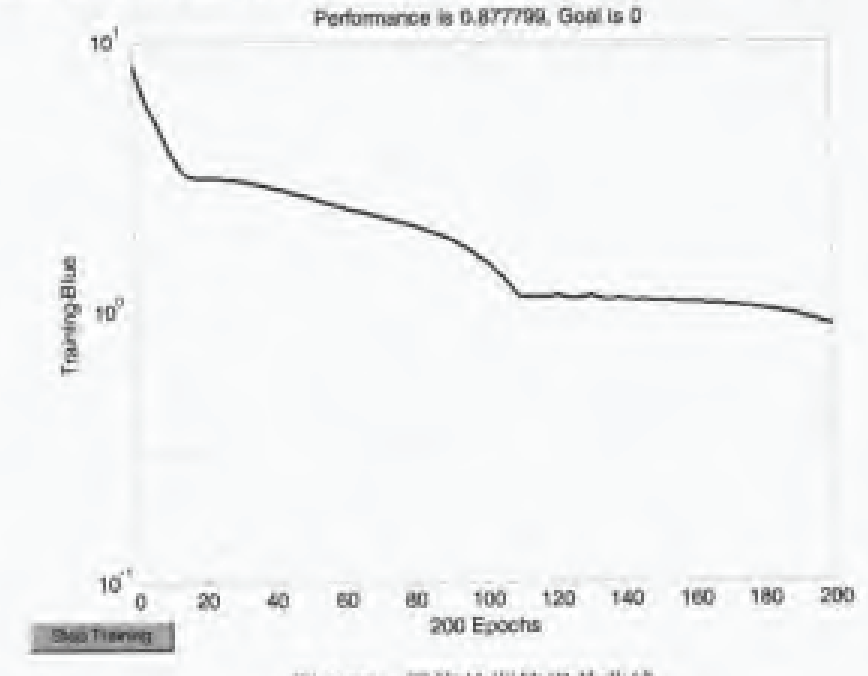


图 2-20　网络的训练误差曲线

此时，网络对 P 和 T 之间非线性关系的拟合情况输出如图 2-21 所示，验证了我们的猜测，网络的性能确实不错。

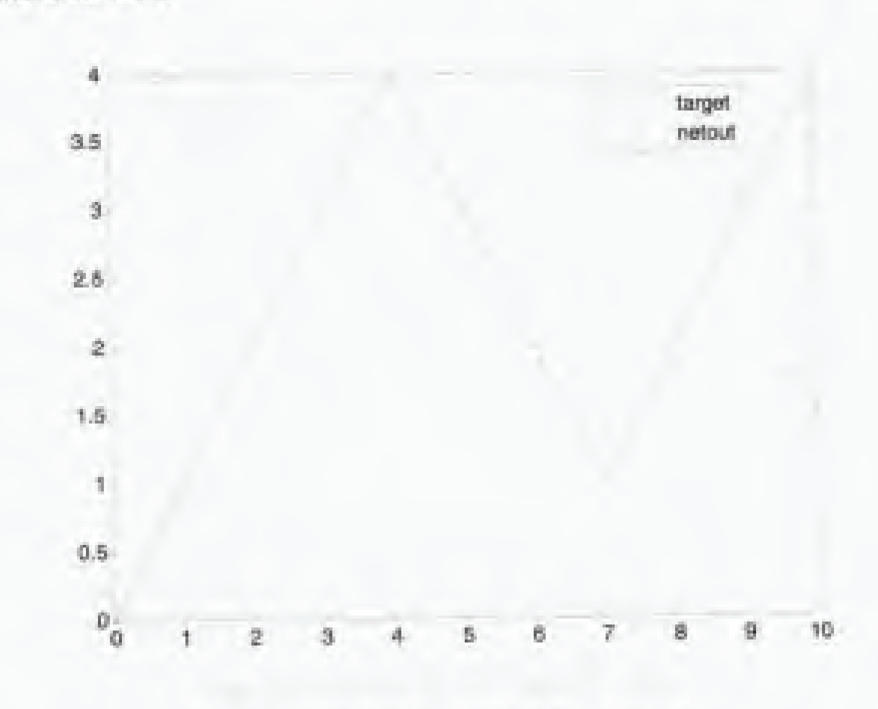


图 2-21　网络的输出（训练函数：traingdx）



利用 newff 创建的前向型网络，每层的权值函数都为 dotprod，输入函数为 netsum，层的初始化函数为 initnw。

2.5 线性网络的神经网络工具箱函数

MATLAB 7 的神经网络工具箱为线性网络提供了大量的函数，它们可分别用于线性网络的设计、创建、分析、训练及仿真等。下面对这些函数的功能、调用格式和应注意的问题等进行详细介绍。

线性网络的常用函数表见 2-8 所示。

表 2-8 线性网络常用函数

函数类型	函数名称	函数用途
线性网络创建函数	newlin	创建一个线性层
	newlind	设计一个线性层
学习函数	learnwh	Widrow-Hoff 学习函数
	maxlinlr	计算线性层的最大学习速率

2.5.1 线性网络创建和设计函数

1）newlin

该函数可以创建一个线性层。所谓线性层是一个单独的层次，它的权函数为 dotprod，输入函数为 netsum，传递函数为 purelin。线性层一般用做信号处理和预测中的自适应滤波器。函数调用格式为：

```
net = newlin
net = newlin(PR,S,ID,LR)
```

其中，

net = newlin：表示在一个对话框中创建一个新的网络；

PR：由 R 个输入元素的最大值和最小值组成的 R×2 维矩阵；

S：输出向量的数目；

ID：输入延迟向量，默认为[0]；

LR：学习速率，默认为 0.01；

net：函数返回值，一个新的线性层。



如果用 0 代替参数 ID，用输入向量的矩阵 P 代替参数 LR，那么函数 newlin(PR,S,0,P)返回的线性层的稳定学习速率对于 P 来说是最大的。

2）newlind

该函数可以设计一个线性层，它通过输入向量和目标向量来计算线性层的权值和阈值。调用格式为：

```
net = newlind
net = newlind(P,T,Pi)
```

其中，

net = newlind：表示在一个对话框中创建一个新的网络；

P：Q 组输入向量组成的 R×Q 维矩阵；

T：Q 组目标分类向量组成的 S×Q 维矩阵；

Pi：初始输入延迟状态的 ID 个单元阵列，每个元素 Pi{i,k}都是一个 Ri×Q 维的矩阵，默认为空。

net：函数返回值，一个线性层，它的输出误差平方和对于输入 P 来说具有最小值。

例 2.11 在输入 P、输出目标 T 和初始输入延迟给定的情况下，利用函数 newlind 创建一个线性层，并对其进行仿真。

MATLAB 代码如下：

```
P = {1 2 1 3 3 2};
Pi = {1 3};  %两个初始输入延迟
T = {5.0 6.1 4.0 6.0 6.9 8.0};
net = newlind(P,T,Pi);
Y = sim(net,P,Pi)
```

运行结果如下：

```
Y =
    [4.9824]    [6.0851]    [4.0189]    [6.0054]    [6.8959]    [8.0122]
```

2.5.2 学习函数

1）learnwh

该函数为 Widrow-Hoff 学习函数，也称为 delta 准则或最小方差准则学习函数。它可以修改神经元的权值和阈值，使输出误差的平方和最小。它沿着误差平方和的下降最快方向连续调整网络的权值和阈值，由于线性网络的误差性能表面是抛物面，仅有一个最小值，因此可以保证网络是收敛的，前提是学习率不超出由函数 maxlinlr 计算得到的最大值。调用格式为：

```
[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learnwh(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnwh(code)
```

其中，

W：S×R 维的加权矩阵（或为 S×1 的阈值矩阵）；

P：Q 个 R 维的输入向量（或为 Q 个单值输入）；

Z：Q 组 S 维的加权输入向量；

N：Q 组 S 维的网络输入向量；

A：Q 组 S 维的输出向量；

T：Q 组 S 维的目标向量；

E：Q 组 S 维的误差向量；

gW：S×R 维的性能参数的梯度；

gA：Q 组 S 维的性能参数的输出梯度；

LP：学习参数，若没有则为空；

LS：学习状态，初始值为空；
dW：S×R 维权值（或阈值）的变化矩阵；
LS：新的学习状态；
learnp(code)：针对不同的 code 返回相应的有用信息，包括：
'pnames' – 返回学习参数的名称；
'pdefaults' – 返回默认的学习参数；
'needg' – 如果函数使用了 gW 或 gA，则返回 1。



函数的学习参数 LP 可以自行设定，如设定学习速率 LP.lr=0.01，这就是其默认值。

2）maxlinlr

该函数为分析函数，用于计算线性层的最大学习速率。调用格式为：

```
lr = maxlinlr(P)
lr = maxlinlr(P,'bias')
```

其中，

P：输入向量的 R×Q 维矩阵；
lr = maxlinlr(P)：针对不带阈值的线性层得到一个所需要的最大学习速率；
lr = maxlinlr(P,'bias')：针对带有阈值的线性层得到一个所需要的最大学习速率。

下面一组代码可在给定输入 P 的情况下，分“带阈值”和“不带阈值”两种情况求得该线性层所需的最大学习速率。

```
P = [1 2 -4 7; 0.1 3 10 6];
lr_bias = maxlinlr(P,'bias')
lr=maxlinlr(P)
```

运行结果为：

```
lr_bias = 0.0067
lr = 0.0069
```



一般来说，学习速率越大，所需的训练时间就越少。但是，如果学习速率过大，则容易造成学习过程的不稳定。

例 2.12 线性网络经常用做信号处理和预测中的自适应滤波器。本例试图创建一个单输入、单神经元的线性网络，输入延迟为 0 和 1，学习速率为 0.01。

```
net = newlin([-1 1],1,[0 1],0.01);
```

接下来针对输入向量 P1 对网络进行仿真。

```
P1 = {0 -1 1 1 0 -1 1 0 0 1};
Y = sim(net,P1);Y
```

输出结果为：

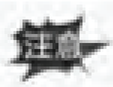
```
Y =
    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]
```

可见，此时的所有输出都为 0。

定义一个目标向量 T1，调用自适应函数 adapt 对网络进行自适应处理。

```
T1 = {0 -1 0 2 1 -1 0 1 0 1};
```

```
[net,Y,E,Pf] = adapt(net,P1,T1); Y
```



由于这是第一次调用函数 adapt，因此需要用到默认的输入延迟条件。

输出结果为：

```
Y =
  Columns 1 through 8
    [0]    [0]    [0]    [0]    [0.0300]    [-0.0103]    [0.0200]    [0.0395]
  Columns 9 through 10
    [0.0192]    [0.0587]
```

经过自适应处理后的网络，输出有了明显的变化。

给出一组新的输入和目标序列，利用上一次自适应的最终条件 Pf 作为本次自适应过程的初始条件，对网络再次进行自适应。

```
P2 = [1 0 -1 -1 1 1 1 0 -1];
T2 = [2 1 -1 -2 0 2 2 1 0];
[net,Y,E,Pf] = adapt(net,P2,T2,Pf); Y
```

输出结果为：

```
Y =
  Columns 1 through 6
    [0.1170]    [0.1056]    [-0.0117]    [-0.0988]    [0.0379]    [0.2101]
  Columns 7 through 9
    [0.2638]    [0.1841]    [-0.0614]
```

最后，利用所有的输入序列和目标序列对网络进行训练，训练次数设定为 200 次，目标误差为 0.1。在训练之前，对网络的权值和阈值进行初始化。

```
net=init(net);
P3 = [P1 P2];
T3 = [T1 T2];
net.trainParam.epochs = 200;
net.trainParam.goal = 0.1;
net = train(net,P3,T3);
Y = sim(net,[P1 P2]);Y
```

网络训练过程的误差曲线如图 2-22 所示，训练结果为：

```
TRAINB, Epoch 0/200, MSE 1.47368/0.1.
TRAINB, Epoch 11/200, MSE 0.0986643/0.1.
TRAINB, Performance goal met.
```

此时的输出结果为：

```
Y =
  Columns 1 through 6
    [0.1869]    [-0.5581]    [0.1638]    [1.7000]    [0.9550]    [-0.5581]
  Columns 7 through 12
    [0.1638]    [0.9550]    [0.1869]    [0.9319]    [1.7000]    [0.9550]
  Columns 13 through 18
    [-0.5581]    [-1.3262]    [0.1638]    [1.7000]    [1.7000]    [0.9550]
  Column 19
```

[-0.5581]

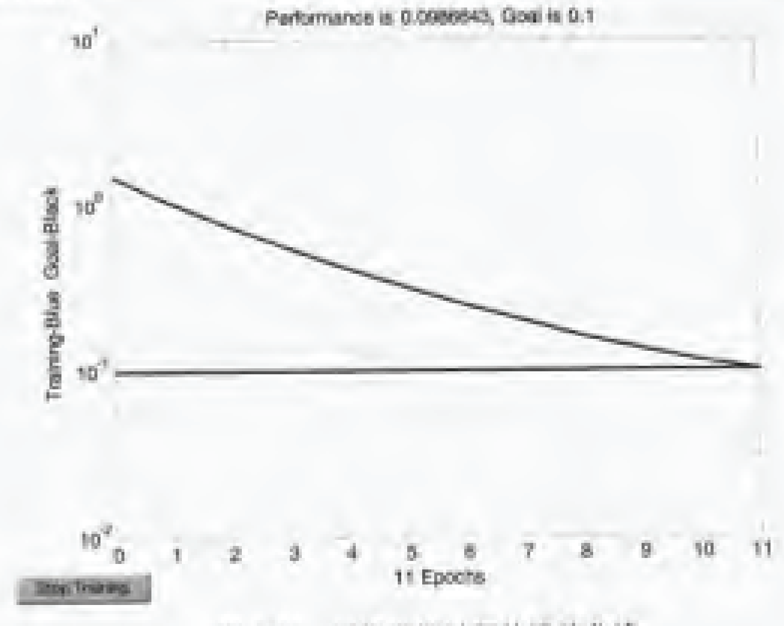


图 2-22 网络训练过程的误差曲线



由 newlin 创建的线性网络：权值函数为 dotprod，输入函数为 netsum，神经元传递函数为 purelin，权值和阈值的初始化函数为 initzero，学习函数为 learnwh，性能函数为 mse。

2.6 自组织竞争网络的神经网络工具箱函数

自组织竞争人工神经网络以无教师教学的方式进行网络训练，具有自组织功能，网络通过自身训练，自动对输入模式进行分类。自组织竞争人工神经网络的基本思想是网络竞争层中的各神经元通过竞争来获取对输入模式的响应机会，最后仅剩一个神经元成为竞争的胜利者，并对那些与获胜神经元有关的各连接全朝着更有利于它竞争的方向调整。因此，自组织竞争人工神经网络进一步拓宽了神经网络在分类和模式识别方面的应用。

MATLAB 7 的神经网络工具箱为自组织竞争网络提供了大量的函数工具。本节将详细介绍这些函数的功能、调用格式和注意事项等。表 2-9 列出了自组织竞争网络常用的函数。

表 2-9 自组织网络常用函数表

函数类型	函数名称	函数用途
网络创建函数	newc	创建一个竞争层
	newsom	创建一个自组织特征映射
	newlvq	创建一个学习向量量化网络
神经元传递函数	compet	竞争性传递函数
	softmax	软最大传递函数

（续表）

函数类型	函数名称	函数用途
距离函数	boxdist	Box 距离函数
	dist	欧氏距离权函数
	linkdist	连接距离函数
	mandist	Manhattan 距离权函数
学习函数	learnk	Kohonen 权值学习函数
	learnsom	自组织映射权值学习函数
	learnis	Instar 权值学习函数
	learnos	Outstar 权值学习函数
初始化函数	midpoint	中点权值初始化函数
权值函数	negdist	负距离权值函数
显示函数	plotsom	绘制自组织特征映射
拓扑函数	hextop	六角层结构函数
	gridtop	网格层结构函数
	randtop	随机层结构函数

2.6.1 神经网络创建函数

1）newc

该函数用于创建一个竞争层。调用格式为：

```
net = newc
net = newc(PR,S,KLR,CLR)
```

其中，

net = newc：表示在对话框中创建一个新的网络；

PR：R 个输入元素的最大值和最小值的设定值，R×2 维矩阵；

S：神经元的数目；

KLR：Kohonen 学习速率，默认为 0.01；

CLR：Conscience 学习速率，默认为 0.001；

net：函数返回值，一个新的竞争层。

2）newsom

该函数用于创建一个自组织特征映射。调用格式为：

```
net = newsom
net = newsom(PR,[d1,d2,...],tfcn,dfcn,olr,osteps,tlr,tns)
```

其中，

net = newsom：表示在对话框中创建一个新的网络；

PR：R 个输入元素的最大值和最小值的设定值，R×2 维矩阵；

di：第 i 层的维数，默认为[5,8]；

tfcn：拓扑函数，默认为"hextop"；

dfcn：距离函数，默认为"linkdist"；

olr：分类阶段学习速率，默认为0.9；

osteps：分类阶段的步长，默认为1000；

tlr：调谐阶段的学习速率，默认为0.02；

tns：调谐阶段的邻域距离，默认为1。

函数返回一个自组织特征映射。

3）newlvq

该函数用于创建一个学习向量量化LVQ网络。调用格式为：

```
net = newlvq
net = newlvq(PR,S1,PC,LR,LF)
```

其中，

net = newlvq：用于在对话框中创建一个LVQ网络；

PR：R×2的矩阵，指定了输入向量中元素的最大值和最小值；

S1：竞争层神经元的数目；

PC：分类的百分比；

LR：学习速率，默认为0.01；

LF：学习函数，默认为learnlv1。



学习函数LF可以选用learnlv1或learnlv2。但是，应用learnlv2是有限制的，只有在利用learnlv1对网络进行训练后，才可以调用learnlv2来结束训练。

2.6.2 传递函数

1）compet

该函数为竞争性传递函数，用于神经元的网络输入转换。调用格式为：

```
A = compet(N)
info = compet(code)
```

其中，

N：输入（列）向量的S×Q维矩阵；

A：函数返回值，输出向量矩阵，每一列向量只有一个1，位于输入向量最大的位置；

info = compet(code)：根据code值的不同返回有关函数的不同信息，包括：

deriv——返回导函数的名称；

name——函数全称；

output——输出范围；

active——动态输入范围。

下面给出一组MATLAB代码，可以演示该函数的功能和运行机理。

```
n = [0; 1; -0.5; 0.5];
a = compet(n)
info=compet('name')
```

```
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

运行结果为：

```
a =(2,1)：1
info =Competitive
```

向量 a 和 n 如图 2-23 所示。

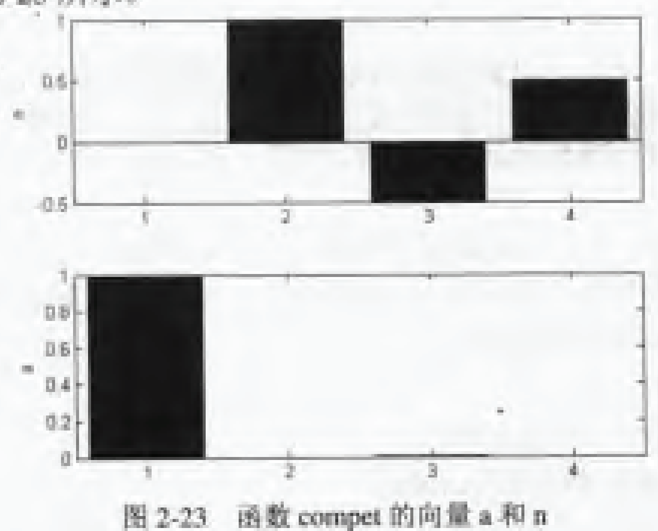


图 2-23　函数 compet 的向量 a 和 n

2）softmax

该函数为软最大传递函数。调用格式为：

```
A = softmax(N)
info = softmax(code)
```

各参数含义请参见 compet。与 compet 不同的是，参数 A 为函数返回向量，各元素在区间[0,1]之间，且向量结构与 N 一致。

利用如下一组代码演示该函数的功能和运行机理：

```
n = [0; 1; -0.5; 0.5];
a = softmax(n)
info=softmax('name')
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

运行结果为：

```
a =
      0.1674
      0.4551
      0.1015
      0.2760
info =
      Soft Max
```

向量 a 和 n 如图 2-24 所示。

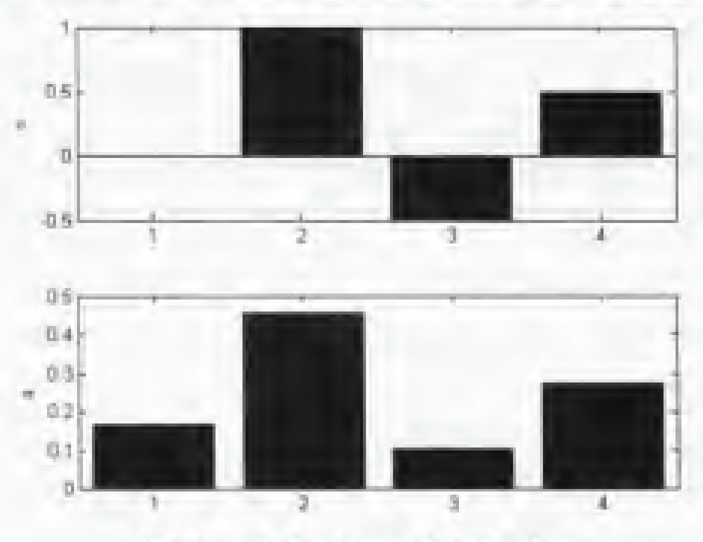


图2-24 函数 softmax 的向量 a 和 n

2.6.3 距离函数

1）boxdist

该函数为 Box 距离函数。在给定神经网络某层的神经元的位置后，可利用该函数计算神经元之间的距离。该函数通常用于结构函数的 gridtop 的神经网络层。调用格式为：

```
d = boxdist(pos)
```

其中，

pos：神经元位置的 N×S 维矩阵；

d：函数返回值，神经元距离的 S×S 维矩阵。

函数的运算原理为 $d(i,j)=\max\|P_i-P_j\|$。其中，$d(i,j)$ 表示距离矩阵中的元素；P_i 表示位置矩阵的第 i 列向量。

以下 MATLAB 代码可以演示该函数的运算规则：

```
pos = rand(3,4)  %产生一个3×4的矩阵
d = boxdist(pos)
```

运行结果为：

```
pos =
    0.9501    0.4860    0.4565    0.4447
    0.2311    0.8913    0.0185    0.6154
    0.6068    0.7621    0.8214    0.7919
d =
         0    0.6602    0.4937    0.5054
    0.6602         0    0.8728    0.2759
    0.4937    0.8728         0    0.5969
    0.5054    0.2759    0.5969         0
```



每次运行上面的代码，矩阵 pos 和 d 的值可能都会不一样，这是由于矩阵 pos 是随机产生的缘故。

2）dist

该函数为欧氏距离权函数，通过对输入进行加权得到加权后的输入。调用格式为：

```
Z = dist(W,P)
df = dist('deriv')
D = dist(pos)
```

其中，

W：S×R 维的权值矩阵；

P：Q 组输入（列）向量的 R×Q 维矩阵；

Z：S×Q 维的距离矩阵；

pos：神经元位置的 N×S 维矩阵；

D：S×S 维的距离矩阵；

df = dist('deriv')：返回值为空，因为该函数不存在导函数。

函数的运算规则为 $D = \mathrm{sqrt}\left(\mathrm{sum}\left((x-y)^2\right)\right)$，其中 x 和 y 分别为列向量。下面用一组代码演示该函数的运行机理：

```
W = rand(4,3);
P = rand(3,1);
pos = rand(3,4);
Z = dist(W,P)
D = dist(pos)
```

运行结果为：

```
Z =
    1.0815
    0.9450
    0.6496
    0.7643
D =
         0    0.4316    0.2280    0.8063
    0.4316         0    0.2340    0.7917
    0.2280    0.2340         0    0.6875
    0.8063    0.7917    0.6875         0
```

3）linkdist

该函数为连接距离函数，在给定神经元的位置后，该函数可用于计算神经元之间的距离。调用格式为：

```
d = linkdist(pos)
```

其中，

pos：N×S 维的神经元位置矩阵；

d：S×S 维的距离矩阵。

函数的运算原理为：

$$d(i,j)\begin{cases}0 & 如果 i=j \\ 1 & 如果 \mathrm{sum}\left(\left(P_i-P_j\right)^2\right)^{\frac{1}{2}}\leqslant 1 \\ 2 & 如果存在 k，使得 d(i,k)=d(k,j)=1 \\ 3 & 如果存在 k_1,k_2，使得 d(i,k_1)=d(k_1,k_2)=d(k_2,j)=1 \\ N & 如果存在 k_1,k_2,\cdots,k_N，使得 d(i,k_1)=d(k_1,k_2)=\cdots=d(k_N,j)=1 \\ S & 其他\end{cases}$$

下面用一组代码演示该函数的运算原理：

```
pos = rand(3,4);
D = linkdist(pos)
```

运行结果为：

```
D =
     0     1     1     1
     1     0     1     1
     1     1     0     1
     1     1     1     0
```

4）mandist

该函数为 Manhattan 距离权函数。调用格式为：

```
Z = mandist(W,P)
df = mandist('deriv')
D = mandist(pos);
```

各参数含义请参见 dist。

函数的运算原理为 $d=\mathrm{sum}\left(\mathrm{abs}\left(X-Y\right)\right)$，其中 X 和 Y 为两个向量。利用下面一组代码进行演示：

```
pos = rand(3,4);
d=mandist(pos)
```

运行结果为：

```
d =
          0    1.5282    1.3646    0.7414
     1.5282         0    1.2715    1.1293
     1.3646    1.2715         0    1.0076
     0.7414    1.1293    1.0076         0
```

2.6.4 学习函数

1）learnk

该函数为 Kohonen 权值学习函数。调用格式为：

```
[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnk(code)
```

其中，

W：S×R 维的权值矩阵（或 S 维的阈值向量）；
P：R×Q 维的输入向量矩阵（也可用 ones(1,Q)产生）；
Z：S×Q 权值输入向量矩阵；
N：S×Q 网络输入向量矩阵；
A：S×Q 输出向量矩阵；
T：S×Q 层目标向量矩阵；
E：S×Q 层误差向量矩阵；
gW：S×R 性能梯度矩阵；
gA：S×Q 输出性能梯度矩阵；
D：S×S 神经元距离矩阵；
LP：学习参数，若无则为空；
LS：学习状态，初始化为空；
dW：S×R 维的权值（阈值）变化矩阵；
LS：新的学习状态；
info = learnk(code)根据不同的 code 值返回不同的相关信息，包括：
pnames——返回学习参数名；
pdefaults——返回默认的学习参数；
needg——如果函数使用了参数 gW 或 gA，则返回 1。

在利用该函数进行学习之前，需要设定学习参数。例如，学习速率的设置格式为 LP.lr = 0.01，实际上这也是学习速率的默认值。

函数利用 Kohonen 规则进行学习。

2）learnsom

该函数为自组织映射权值学习函数。调用格式为：

```
[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnsom(code)
```

各参数含义请参见 learnk。

在利用该函数进行学习之前，需要设置以下学习参数，如表 2-10 所示。

表 2-10　学习参数

函数名称	默认值	属　　性
LP.order_lr	0.9	分类阶段学习速率
LP.order_steps	1000	学习阶段步长
LP.tune_lr	0.02	调谐阶段邻域距离
LP.tune_nd	1	调谐阶段学习速率

以上各值都是 MATLAB 的默认值，如需调整，只要按照上面的格式重新进行设定即可。

3）learnis

该函数为 Instar 权值学习函数。调用格式为：

```
[dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnis(code)
```

各参数的含义请参见 learnk。

使用该函数前，需要设置学习速率，如果 LP.lr = 0.01，这就是 MATLAB 的默认值。如需调整，只要做相应改动即可。

4）learnos

该函数为 Outstar 权值学习函数。调用格式为：

```
[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
```

各参数的含义和学习参数的设定，请参见 learnis。

2.6.5 初始化函数

1）midpoint

该函数为中点权值初始化函数。调用格式为：

```
W = midpoint(S,PR)
```

其中，

S：神经元的数目；

PR：每组输入向量的最大值和最小值组成的 R×2 维矩阵，规定了输入区间为[Pmin Pmax]；

W：函数返回值，S×R 维的矩阵，每个元素对应设定为(Pmin+Pmax)/2。

运行下面的代码，可知该函数的运行原理：

```
W = midpoint(5,[0 1; -2 2])
```

运行结果为：

```
W =
    0.5000    0
    0.5000    0
    0.5000    0
    0.5000    0
    0.5000    0
```

这句代码中的矩阵[0 1; -2 2]设定了输入的范围；5 表示神经元的数目是 5 个。

2.6.6 权值函数

1）negdist

该函数为负距离权值函数。调用格式为：

```
Z = negdist(W,P)
df = negdist('deriv')
```

其中，

W：S×R 维的权值矩阵；

P：Q 组输入向量的 R×Q 维矩阵；

df = negdist('deriv')：返回值为空，因为该函数不存在导函数。

该函数的运算原理为：

$$z = -\text{sqrt}(\text{sum}(w - p)^2)$$

实际上这也是负欧氏距离。

利用下面一组代码演示该函数的运行机理：

```
W = rand(4,3);  %产生一个4×3维的权值矩阵
P = rand(3,1);  %产生一个3维的输入向量
Z = negdist(W,P)
```

运行结果为：

```
Z =
    -0.6637
    -0.7414
    -0.6095
    -1.0426
```

2.6.7 显示函数

1）plotsom

该函数用于绘制自组织特征映射。调用格式为：

```
plotsom(pos)
plotsom(W,D,ND)
```

其中，

pos：S 个 N 维神经元的位置向量；

W：权值矩阵；

D：距离矩阵；

ND：邻域矩阵，默认为 1。

plotsom(pos)利用红点绘制神经元的位置，将欧氏距离小于等于 1 的神经元连接起来；plotsom(W,D,ND)将欧氏距离小于等于 1 的神经元的权值向量连接起来。

2.6.8 结构函数

1）hextop

该函数为六角层结构函数。调用格式为：

```
pos = hextop(dim1,dim2,...,dimN)
```

其中，

dimi：维数为 i 的层的长度；

pos：由 N 个并列向量组成的 N×S 维矩阵，其中，S=dim1×dim2×…×dimN。

可以利用该函数创建一个二维的神经网络层，共有 40 个神经元，分布在一个 8×5 的六角晶格上。代码如下：

```
pos = hextop(8,5);
plotsom(pos)
```

运行结果如图 2-25 所示。

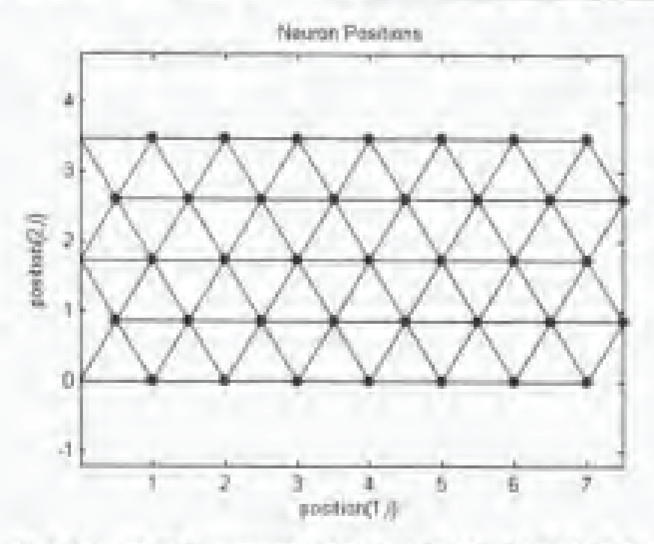


图 2-25　函数 hextop 产生的 40 个神经元的分布位置

2）gridtop

该函数为网格层结构函数。调用格式为：

```
pos = gridtop(dim1,dim2,...,dimN)
```

各参数含义请参见 hextop。

同样，可以利用该函数创建一个二维的神经网络层，共有 40 个神经元，分布在一个 8×5 的网格上。代码如下：

```
pos =gridtop(8,5);
plotsom(pos)
```

运行结果如图 2-26 所示。

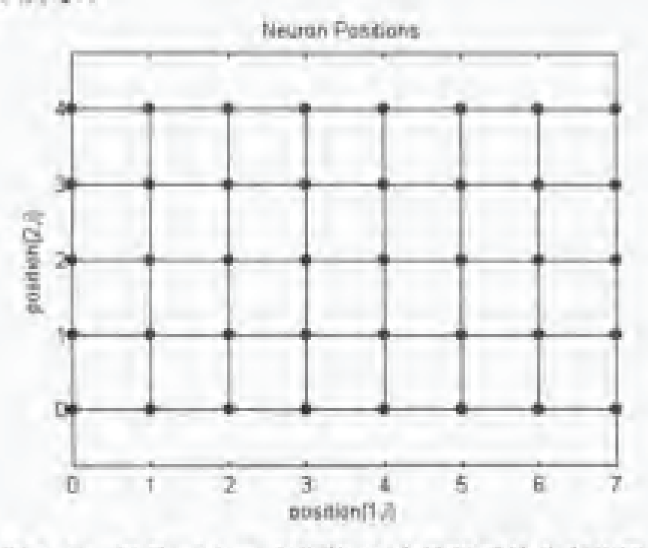


图 2-26　函数 gridtop 产生的 40 个神经元的分布位置

3）randtop

该函数为随机层结构函数。调用格式为：

```
pos = randtop(dim1,dim2,...,dimN)
```

各参数含义请参见 hextop。

同样，可以利用该函数创建一个二维的神经网络层，共有 40 个神经元，分布在一个 8

×5 的随机格上。代码如下：

```
pos = randtop(8,5);
plotsom(pos)
```

运行结果如图 2-27 所示。

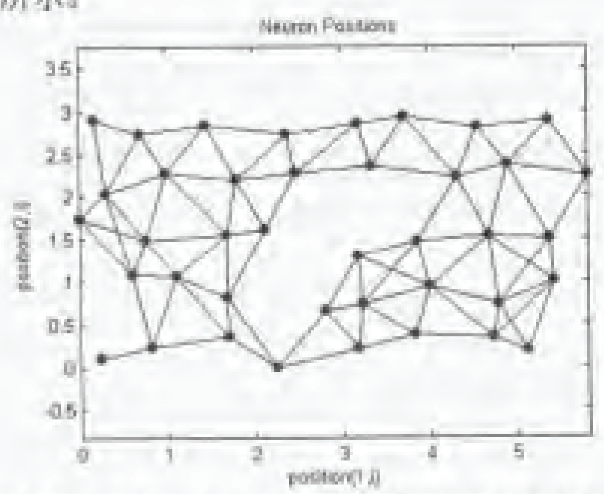


图 2-27 函数 randtop 产生的 40 个神经元的分布位置

例 2.13 基本竞争型网络特别适合于解决模式分类问题。本例给定一组输入向量 P，建立一个基本竞争型网络对其进行分类。

```
P = [0.1 0.8 0.1 0.9; 0.2 0.9 0.1 0.8];
```

可见，输入向量应该分为两组，一组为比较小的点，另一组为比较大的点，如图 2-28 所示。由图可见，左下角的两个点为一组，右上角的两个点为一组。

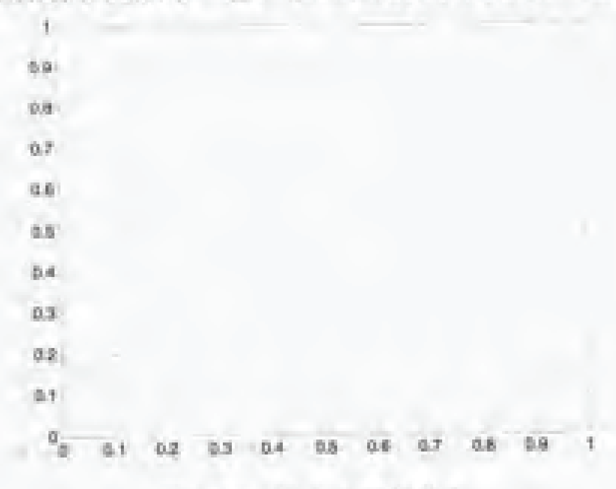


图 2-28 输入向量的分布

接下来创建一个基本竞争型网络，对其进行分类。

```
net = newc([0 1; 0 1],2);
net = train(net,P);
Y = sim(net,P)
Yc = vec2ind(Y)
```

由于输入向量被分为两类，因此，竞争层有两个神经元。网络的训练结果为：

```
TRAINR, Epoch 0/100
TRAINR, Epoch 25/100
```

```
TRAINR, Epoch 50/100
TRAINR, Epoch 75/100
TRAINR, Epoch 100/100
TRAINR, Maximum epoch reached.
```

训练好的网络输出为：

```
Y =
   (1,1)        1
   (2,2)        1
   (1,3)        1
   (2,4)        1
Yc =
     1     2     1     2
```

可见，网络确实将输入向量分为两类，一类为（0.1 0.2）、（0.1 0.1）两个点，剩下的两组为第二类。



通过 newc 创建的基本竞争型神经网络只有一个竞争层，权值函数为 negdist，输入函数为 netsum，传递函数为 compet，初始化函数为 midpoint 或 initcon，训练函数或自适应函数为 trains 和 trainr，学习函数为 learnk 或 learncon。

例 2.14 针对上例中的输入向量 P，建立一个自组织特征映射，并对 P 进行分类。

首先建立一个 SOM 网络，并绘制出网络当前神经元的位置。

```
P = [0.1 0.8 0.1 0.9; 0.2 0.9 0.1 0.8];
net = newsom([0 2; 0 1],[3 5]);
plotsom(net.layers{1}.positions)
```

创建的 SOM 网络的竞争层为一个二维的 3×5 的平面阵列，拓扑函数为 hextop，距离函数为 linkdist，其他参数均取默认值。

此时的神经元分布如图 2-29 所示。由图可见，此时的神经元位置是均匀分布的，也就是说，网络还没有对输入向量进行分类的能力。

接下来对网络进行训练，检查训练过程中神经元位置的变化，分别如图 2-30、图 2-31 所示。

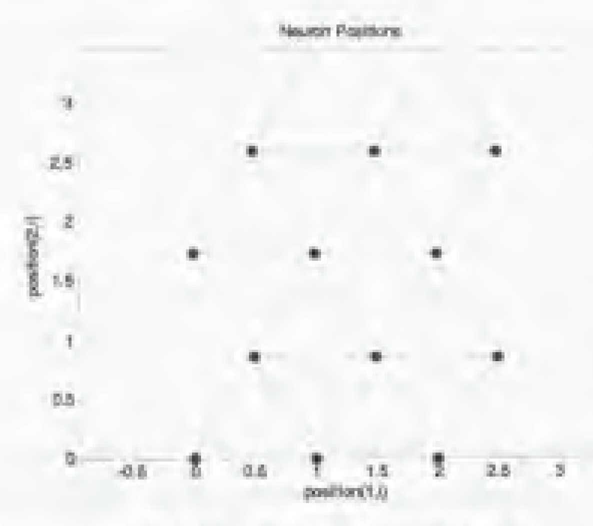


图 2-29 神经元位置的初始分布

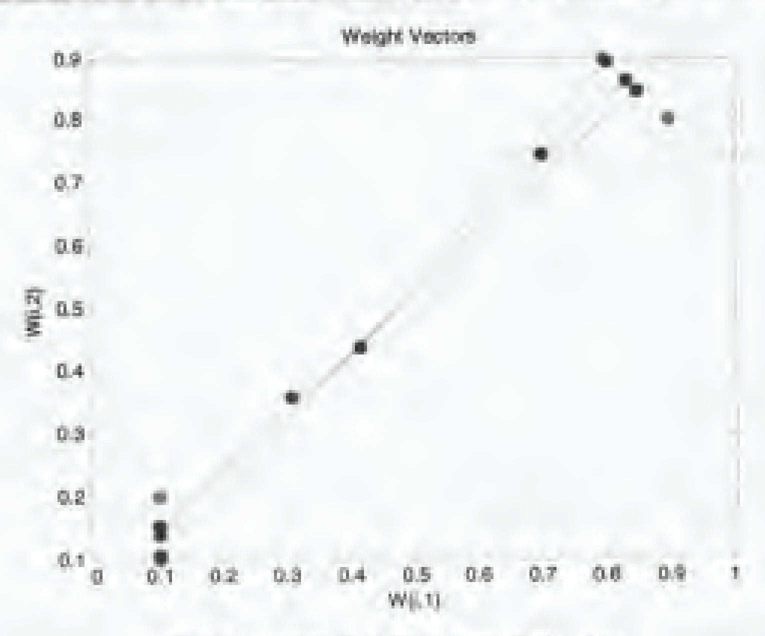


图 2-30　神经元位置（训练次数：10）

由图 2-30 可见，经过 10 次训练后，神经元的位置就发生了明显的改变。神经元位置的分布状况表示它们已经可以对输入向量进行分类了。此时，再增加训练次数已经没有什么实际意义了，经过 25 次训练后的网络神经元分布和 10 次训练后的神经元分布没有什么明显的差异，如图 2-31 所示。

经过 10 次训练后，网络的输出为：

```
yc =
      1    13    2    14
```

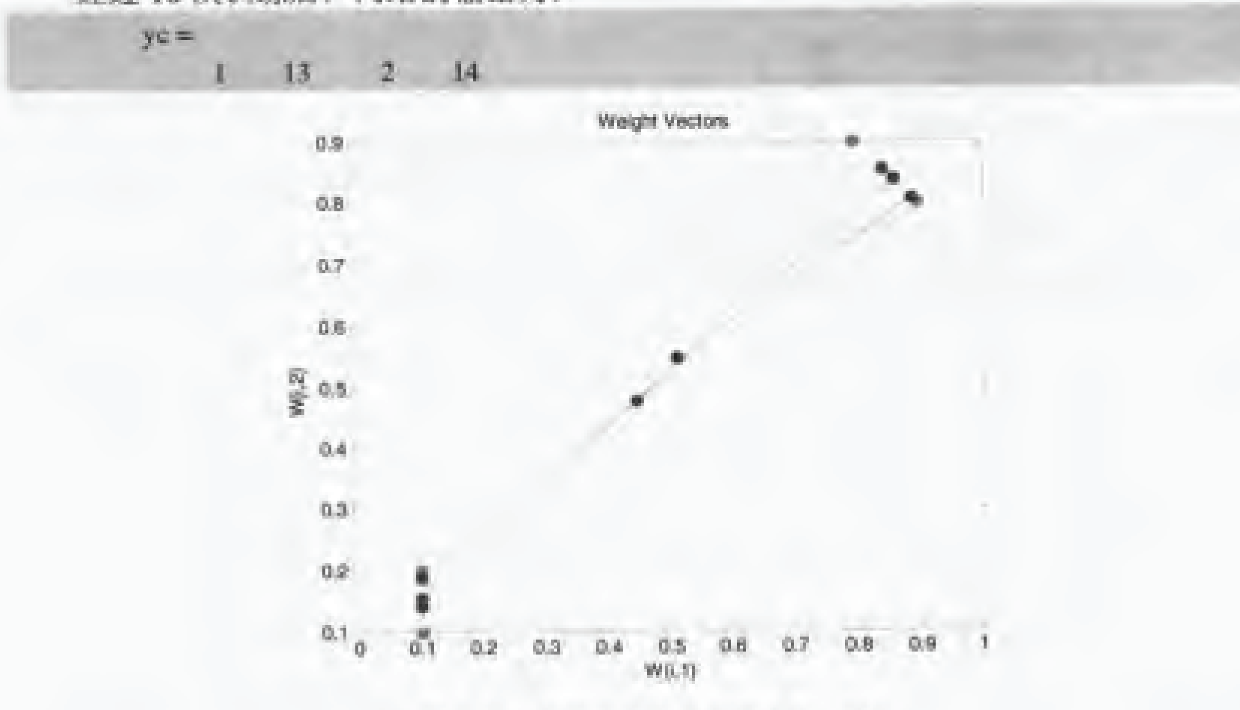


图 2-31　神经元位置（训练次数：25）

由此可见，SOM 网络的分类结果比较精细，把 4 个点分为了 4 类，即激发了 1、13、2、14 四个神经元。但是，通过对结果进行分析可以发现，在上例中处于同一类的点在这里激发的神经元位置也是邻近的，这说明相对于基本竞争型网络来说，SOM 网络的分类结

果更加准确。

经过 25 次训练后，网络的输出值与此类似。需要注意的是，重新运行上述代码，可能结果就会不一致，这是因为每次激发的神经元不一样。但是，相似的类激发的神经元总是邻近的，差别很大的类激发的神经元相差也比较远。



利用基本竞争型网络进行分类，需要首先设定输入向量的类别总数，再由此确定神经元的个数。但利用 SOM 网络进行分类却不需要这样。SOM 网络会自动将差别很小的点归为一类，差别不大的点激发的神经元位置也是邻近的。

本例的 MATLAB 代码为：

```
P = [0.1 0.8 0.1 0.9; 0.2 0.9 0.1 0.8];
net = newsom([0 2; 0 1],[3 5]);
plotsom(net.layers{1}.positions);
%进行训练
%训练次数为 10
net.trainParam.epochs = 10;
net = train(net,P);
plot(P(1,:),P(2,:),'.g','markersize',20);
hold on;
%绘制训练后神经元的位置
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off;
figure;
%训练次数为 25
%训练前进行初始化
net=init(net);
net.trainParam.epochs = 20;
net = train(net,P);
plot(P(1,:),P(2,:),'.g','markersize',20);
hold on;
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off;
```

例 2.15 同样地，输入向量 P，创建一个 LVQ 网络，对 P 进行分类。

假定 P 的分类结果为：

```
Tc=[1 2 1 2];
```

接下来建立一个 LVQ 网络，并对其进行训练。

```
T = ind2vec(Tc);
net = newlvq(minmax(P),4,[.6 .4]);
net = train(net,P,T);
```

训练过程中的误差曲线如图 2-32 所示。经过 2 次训练后，网络的输出误差满足性能要求。

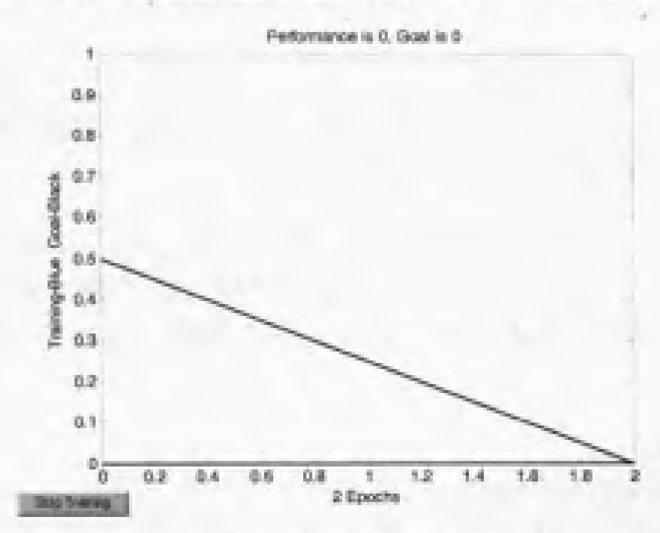


图 2-32　训练误差曲线

对网络进行仿真：

```
Y = sim(net,P)
Yc = vec2ind(Y)
```

结果为：

```
Y =
        1     0     1     0
        0     1     0     1
Yc =
        1     2     1     2
```

可见网络的训练和分类是成功的。输入一组新的点 P_test，检查网络的分类功能。

```
P_test=[0.2 0.3;0.7 0.9];
y_test=sim(net,P_test);
yc_test=vec2ind(y_test)
```

结果为：

```
yc_test =
        1     2
```

网络将输入向量分为两类，第一类为（0.2 0.3），第二类为（0.7 0.9）。这与实际的分类情况是一致的，可见网络的分类性能比较好。

本例的 MATLAB 代码为：

```
P = [0.1 0.8 0.1 0.9; 0.2 0.9 0.1 0.8];
Tc=[1 2 1 2];
T = ind2vec(Tc);
net = newlvq(minmax(P),4,[.6 .4]);
net = train(net,P,T);
Y = sim(net,P)
Yc = vec2ind(Y)
P_test=[0.2 0.3;0.7 0.9];
y_test=sim(net,P_test);
yc_test=vec2ind(y_test)
```

2.7 径向基网络的神经网络工具箱函数

径向基（RBF）网络是以函数逼近理论为基础构造的一类前向网络，是由 Powell M.J.D 于 1985 年提出的。由于它具有结构自适应确定、输出与初始权值无关的优良特性，在多维曲面拟合、自由曲面重构和大型设备故障诊断等领域有着比较多的应用。

MATLAB 7 的神经网络工具箱为径向基网络提供了很多工具箱函数，它们对我们利用 MATLAB 进行径向基网络的设计、分析及实际应用有着不可替代的作用。本节将详细介绍这些函数。表 2-11 列出了 RBF 网络常用的函数。

表 2-11 RBF 网络常用函数表

函数类型	函数名称	函数用途
网络创建函数	newrb	创建一个 RBF 网络
	newrbe	创建一个准确的 RBF 网络
	newpnn	创建一个概率神经网络
	newgrnn	设计一个广义回归神经网络
神经元传递函数	radbs	径向基传递函数
转换函数	ind2vec	将数据索引转换为向量组
	vec2ind	ind2vec 的逆函数

2.7.1 神经网络创建函数

1）newrb

该函数可以用来设计一个径向基网络。调用格式为：

```
net = newrb
[net,tr] = newrb(P,T,GOAL,SPREAD,MN,DF)
```

其中，

P：Q 组输入向量组成的 R×Q 维矩阵；

T：Q 组目标分类向量组成的 S×Q 维矩阵；

GOAL：均方误差，默认为 0；

SPREAD：径向基函数的扩展速度，默认为 1；

MN：神经元的最大数目，默认为 Q；

DF：两次显示之间所添加的神经元数目，默认为 25；

net：返回值，一个径向基网络；

tr：返回值，训练记录。



该函数设计的径向基网络 net 可用于函数逼近。径向基函数的扩展速度 SPREAD 越大，函数的拟合就越平滑。但是，过大的 SPREAD 意味着需要非常多的神经元以适应函数的快速变化。如果 SPREAD 设定过小，则意味着需要许多神经元来适应函数的缓慢变化，这样一来，设计的网络性能就不会很好。因此，在网络设计过程中，需要用不同的 SPREAD 值进行尝试，以确定一个最优值。

2）newrbe

该函数用于设计一个准确的径向基网络。调用格式为：

```
net = newrbe
net = newrbe(P,T,SPREAD)
```

各参数含义请参见 newrb。

一般来讲，newrbe 和 newrb 一样，神经元数目越大，对函数的拟合就越平滑。但是，过多的神经元可能会导致计算困难问题。



和 newrb 不同，newrbe 能够基于设计向量快速地、无误差地设计一个径向基网络。

3）newpnn

该函数可用于创建概率神经网络，概率神经网络是一种适用于分类问题的径向基网络。调用格式为：

```
net = newpnn
net = newpnn(P,T,SPREAD)
```

各参数含义请参见 newrbe。



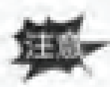
如果 SPREAD 值接近于 0，则创建的概率神经网络可以作为一个最近邻域分类器。随着 SPREAD 值的增大，需要更多地考虑该网络附近的设计向量。

4）newgrnn

该函数可用于设计一个广义回归神经网络。广义回归神经网络是径向基网络的一种，通常用于函数逼近。调用格式为：

```
net = newgrnn
net = newgrnn(P,T,SPREAD)
```

各参数的含义请参见 newrbe。



同 newrb 一样，SPREAD 的值越大，由此设计的网络对函数的拟合就越平滑。为了更精确地对数据进行拟合，最好使 SPREAD 的值小于输入向量之间的典型距离。

2.7.2 转换函数

1）ind2vec

该函数用于将数据索引转换为向量组。调用格式为：

```
vec = ind2vec(ind)
```

其中，

ind：数据索引列向量；

vec：函数返回值，一个稀疏矩阵，每行只有一个 1，矩阵的行数等于数据索引的个数，列数等于数据索引中的最大值。

可通过下面的一组代码演示 ind2vec 的计算原理。

```
ind = [1 3 2 3];
```

```
vec = ind2vec(ind)
```

上述代码的第一行定义了一个数据索引列向量，第二行将其转换为向量组。运行结果为：

```
vec =
    (1,1)        1
    (3,2)        1
    (2,3)        1
    (3,4)        1
```

可以看出，结果应该是一个 4×3 的矩阵，其中（x,y）是与数据索引列向量相对应的。x 为数据索引值，y 表示 x 在数据索引中的位置，由此组成了输出矩阵。矩阵中没有填满的部分需要用 0 补齐。

2）vec2ind

该函数用于将向量组转换为数据索引，与 ind2vec 是互逆的。

2.7.3　传递函数

1）radbas

该函数为径向基传递函数。调用格式为：

```
A = radbas(N)
info = radbas(code)
```

其中，

N：输入（列）向量的 S×Q 维矩阵；

A：函数返回矩阵，与 N 一一对应，即 N 中的每个元素通过径向基函数得到 A；

info = radbas(code)：根据 code 值的不同返回有关函数的不同信息，包括：

deriv——返回导函数的名称；

name——返回函数全称；

output——返回输入范围；

active——返回可用输入范围。

下面的代码可以绘制 radbas 的函数图，如图 2-33 所示。

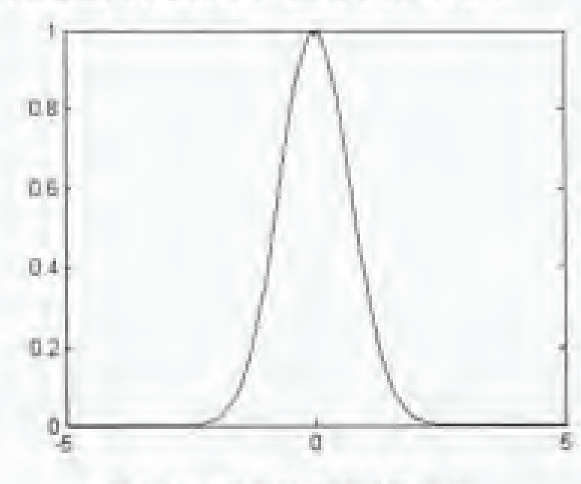


图 2-33　径向基传递函数 radbas

```
n = -5:0.1:5;
a = radbas(n);
plot(n,a)
```



该函数的原型为 $a=\exp\left(-n^2\right)$。

例 2.16 RBF 网络特别适用于解决函数逼近问题，本例利用函数 newrb 创建一个 RBF 网络，并对一个非线性函数 y=sqrt(x)进行逼近。

newrb 创建 RBF 网络是一个不断尝试的过程，在创建过程中，不断增加中间层神经元的个数，直到网络的输出误差满足预先设定的值为止。

```
x=0:0.1:5;
y=sqrt(x);
net=newrb(x,y,0,0.5,20,15);
```

通过上述代码，我们创建了一个目标误差为 0、径向基函数分布密度为 0.5、中间层神经元个数最大值为 20、显示间隔为 5 的 RBF 网络。

代码运行结果为：

```
NEWRB, neurons = 0, SSE = 9.17903
NEWRB, neurons = 5, SSE = 0.924825
NEWRB, neurons = 10, SSE = 0.0394536
NEWRB, neurons = 15, SSE = 0.00184102
NEWRB, neurons = 20, SSE = 3.75484e-005
```

由此可见，当中间层神经元个数增至 20 时，网络输出的误差 SSE 已经非常小了，误差曲线如图 2-34 所示。

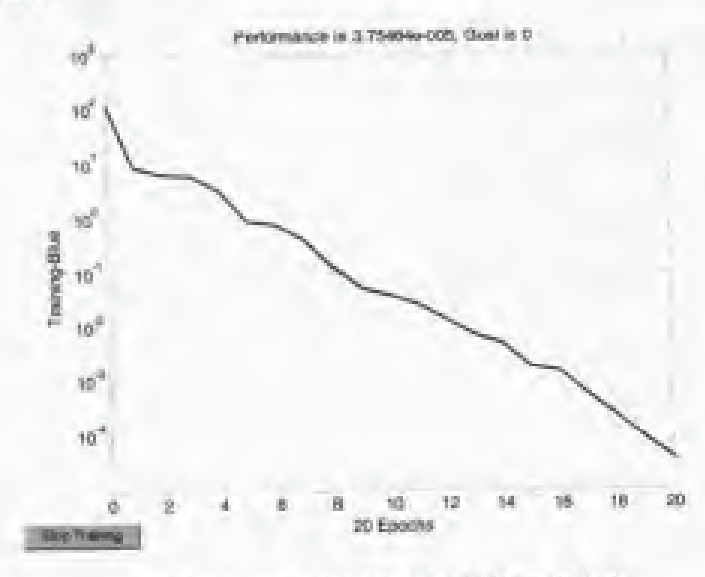


图 2-34 RBF 网络建立过程的误差曲线

利用以下代码得到网络的输出及网络的逼近误差曲线，如图 2-35 所示。

```
t=sim(net,x);
plot(x,y-t,'+-')
```

由图 2-35 可见，网络的逼近误差是非常小的。

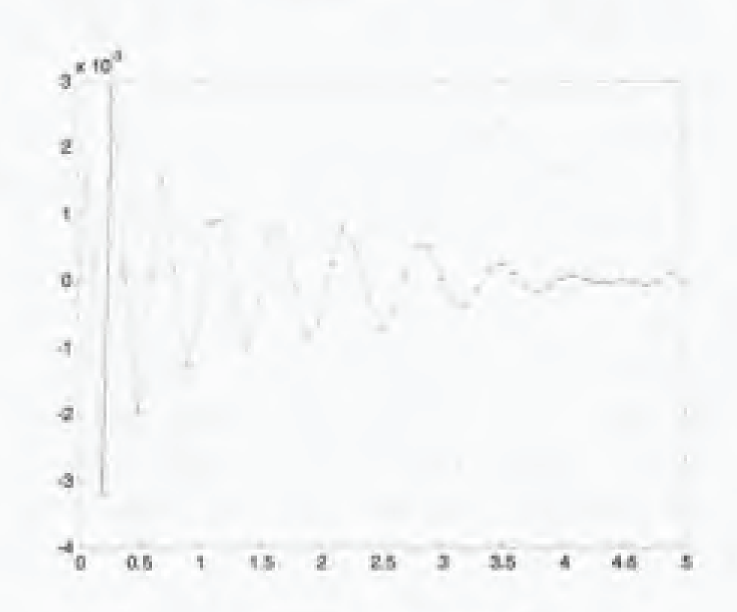


图 2-35 网络的逼近误差曲线

接下来检验网络的外推性能。利用一组样本以外的数据，求网络的仿真输出，并绘制网络的误差曲线。

```
x1=5:0.1:9;
y1=sqrt(x1);
t1=sim(net,x1);
plot(x1,y1-t1,'*-')
```

可见，网络的外推样本为 5~9 之间的采样点，采样周期为 0.1，网络的外推误差曲线如图 2-36 所示。

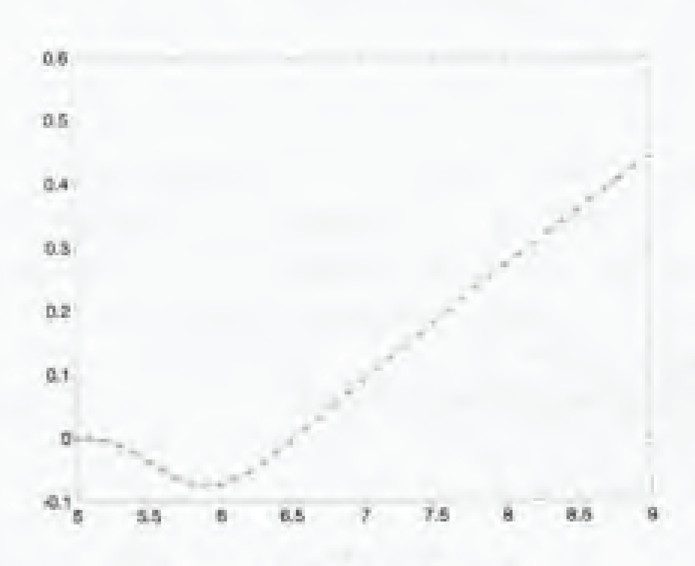


图 2-36 网络的外推误差曲线

网络的外推误差比较高，这是因为训练样本相对比较集中，而外推样本和训练样本之间的距离比较大。仔细观察误差曲线，会发现 5~7.5 之间的样本点的外推误差相对比较小。

2.8 反馈网络的神经网络工具箱函数

反馈神经网络（Recurrent Network）又称为联想记忆网络。其目的是为了设计一个网络，储存一组平衡点，使得在给定一组初始值时，网络通过自行运行能够最终收敛到设计的平衡点上。反馈网络有很多种，其中比较典型的一种是由美国物理学家 Hopfield 提出的 Hopfield 网络。Hopfield 网络的应用主要集中于这些领域：图像和语音处理、数据查询、容错控制、模式分类和识别等。

Elman 网络也是一种反馈网络。它在 BP 网络基本结构的基础上，通过存储内部状态使其具备映射动态特征的功能，从而使系统具有适应时变特性的能力。

MATLAB 7 的神经网络工具箱为 Hofield 网络和 Elman 网络提供了一些工具函数，我们可以借助这些函数，在实际工作中使用 Hopfield 网络和 Elman 网络。

反馈网络常用的工具箱函数见表 2-12。

表 2-12 反馈网络常用函数表

函数类型	函数名称	函数用途
网络创建函数	newhop	设计一个 Hopfield 网络
	newelm	设计一个 Elman 网络
传递函数	satlins	Hopfield 网络的传递函数

2.8.1 Hopfield 网络的工具箱函数

1）newhop

该函数用于设计一个 Hopfield 网络。调用格式为：

```
net = newhop
net = newhop(T)
```

其中，

net = newhop：用于在对话框中创建一个 Hopfield 网络；

T：Q 个目标向量组成的 R×Q 维矩阵，矩阵元素必须为 1 或 -1；

net：函数返回值，创建的 Hopfield 网络，稳定点位于目标向量 T 中。

下面一组代码可创建一个 Hopfield 网络并进行仿真。

```
T = [-1 -1 1; 1 -1 1]';   %矩阵转置
net = newhop(T);   %创建一个 Hopfield 网络
Ai = {[-0.9; -0.8; 0.7]};
[Y,Pf,Af] = sim(net,{1 5},{},Ai);
Y{1}
```

运行结果为 Y{1}=[-1,-1,1]'。由于 Y{1}=T(:,1)，可以看出，设计的 Hopfield 网络已经成功地将存在偏差的向量 Ai 转换为最接近的目标向量 T。

2）satlins

该函数为饱和线性传递函数，通常用作 Hopfield 网络的传递函数。调用格式为：

```
A = satlins(N)
```

```
info = satlins(code)
```

其中，

N：输入（列）向量的 S×Q 维矩阵；

A：函数返回值，限制在区间[-1, 1]中；

info = satlins(code)：根据不同的 code 值返回不同的有关函数的信息，包括：

deriv——返回导函数的名称；

name——返回函数全称；

output——返回输出范围；

active——返回可用输入范围。

函数的原型函数为：

$$\text{satlins}(n)=\begin{cases}-1 & n\leqslant -1\\ n & -1\leqslant n\leqslant 1\\ 1 & n\geqslant 1\end{cases}$$

利用如下的代码可以绘制 satlins 函数，如图 2-37 所示。

```
n = -10:0.1:10;
a = satlins(n);
plot(n,a)
```

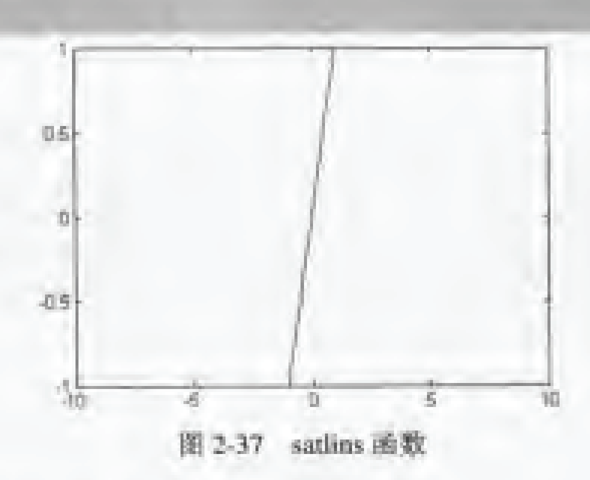


图 2-37　satlins 函数

2.8.2　Elman 网络的工具箱函数

1）newelm

该函数用于设计一个 Elman 网络。调用格式为：

```
net = newelm
net = newelm(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

其中，

net = newelm：用于在对话框中创建一个 Elman 网络；

PR：R 组输入元素的最小值和最大值的设定值，R×2 维的矩阵；

Si：第 i 层的长度；

TFi：第 i 层的传递函数，默认为“tansig”；

BTF：BP 网络的训练函数，默认为"traingdx"；

BLF：BP 网络的权值/阈值学习函数，默认为"learngdm"；

PF：性能函数，默认为"mse"。

参数 BTF 可以取 traingd、traingdm、traingda 和 traingdx 等训练函数；BLF 可以取学习函数 earngd 和 learngdm；PF 可以取性能函数 mse 和 msereg。这些函数在前面章节中都已有所介绍，在此不再赘述。

例 2.17 创建并训练一个 Elman 网络，并评估其性能。

MATLAB 代码如下：

```
%函数 round 将随机数转换为与其最接近的整数
P = round(rand(1,20));
%[]中的后半部分为判断语句，返回 0 或 1
T = [0 (P(1:end-1)+P(2:end) == 2)];
%函数 con2seq 将并发向量转换为序贯向量
Pseq = con2seq(P);
Tseq = con2seq(T);
%网络第一层传递函数为 tansig，第二层为 logsig，训练函数默认为 traingdx
net = newelm([0 1],[10 1],{'tansig','logsig'});
%训练的最大步数为默认值 100
net = train(net,Pseq,Tseq);
Y = sim(net,Pseq)
```

运行结果为：

```
TRAINGDX, Epoch 0/100, MSE 0.423866/0, Gradient 0.3557/1e-006
TRAINGDX, Epoch 25/100, MSE 0.359561/0, Gradient 0.369702/1e-006
TRAINGDX, Epoch 50/100, MSE 0.231497/0, Gradient 0.186045/1e-006
TRAINGDX, Epoch 75/100, MSE 0.196793/0, Gradient 0.0497879/1e-006
TRAINGDX, Epoch 100/100, MSE 0.185103/0, Gradient 0.0341902/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.
```

其中，结果中的前 5 行分别表示训练在进行了 n 步后，以均方误差函数 mse 评估的性能与梯度。最后一行表示训练已经进行了 100 步，达到了最大步数，训练停止，但性能目标没有达到。在这种情况下，如果认为网络的训练误差还不满足要求，可以继续训练。网络训练过程中的误差曲线如图 2-38 所示。

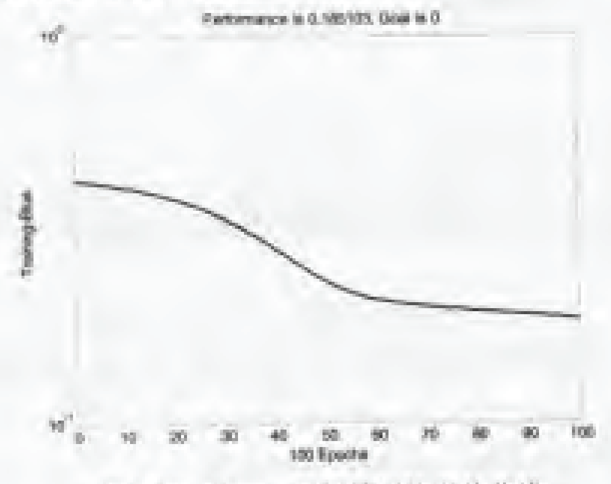


图 2-38　Elman 网络的训练误差曲线



利用函数 newelm 创建的 Elman 网络：权值函数采用 dotprod，输入函数采用 netsum，权值和阈值初始化函数采用 initnw，训练函数采用 trains，而传递函数和学习函数需要特别指定。

例 2.18 因 Hopfield 网络特别适合于模式回想，则本例通过给定一个稳定点 T 来创建一个 Hopfield 网络。

```
T = [-1 -1 1; 1 -1 1]';
net = newhop(T);
```

接下来将稳定点 T 作为网络的初始层延迟条件来检验 T 是否真的是稳定点，如果网络稳定在这些点上，那么输出结果 Y 应该等于 T。

```
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai);Y
```

结果为：

```
Y =
    -1    1
    -1   -1
     1    1
```

Y=T，因此，网络的稳定点确实是 T。



由于 Hopfield 网络不存在输入向量，因此当函数 sim 的第二个参数采用矩阵形式时，Q=2；当采用单元阵列的形式时，{Q TS}=[1 5]。

最后利用一组被污染的向量，通过以上创建的 Hopfield 网络将其恢复出来，网络的仿真步数设定为 5 次。

```
Ai = {[-0.9; -0.8; 0.7]};
[Y,Pf,Af] = sim(net,{1 5},{},Ai);Y{1}
```

结果为：

```
ans =
        -1
        -1
         1
```

由此可见，Ai 恢复到与 T 的第一列元素一致了，这说明网络的回想性能是不错的。Hopfield 网络的这一功能，特别适合于图像补正、自动目标识别等领域。



利用 newhop 创建的 Hopfield 网络：权值函数采用 dotprod，输入函数采用 netsum，传递函数采用 stalins，而网络的权值是自身反馈的。

2.9 小　结

本章首先介绍了一些重要的神经网络工具箱通用函数，对它们的函数格式、调用方法和注意事项等进行了详细的说明。按照这种格式，接下来还分别介绍了感知器、BP 网络、

线性网络、自组织竞争网络、径向基网络和反馈网络的神经网络工具箱函数。读者看完本章后，对神经网络工具箱应该有了初步的掌握，可以利用某些函数来解决一些比较简单的实际问题，但要想全面掌握利用神经网络工具箱来解决问题的过程，还要接着往下看。

第 3 章　前向型神经网络理论及 MATLAB 实现

前向型神经网络结构是层阶型的，信息值可以从输入层单元传播到它上一层的单元。第一层的单元与第二层所有的单元相连，第二层又与其上一层单元相连，同一层中的各单元之间没有连接。前向网络中神经元的输入输出关系，可采用线性阈值硬变换或单元上升的非线性变换，它们的权算法都是采用有教师的 δ 学习律。根据神经元输入输出传递函数的差异、学习算法和网络结构上的某些区别，可将前向型神经网络分为感知器网络、BP 网络、线性网络、径向基网络以及 GMDH 网络等不同的网络模型。本章将针对以上网络类型，对其理论基础进行简单说明，并重点介绍如何利用 MATLAB 7 神经网络工具箱实现这些网络。

本章主要内容：

- *感知器网络及 MATLAB 实现*
- *BP 网络及 MATLAB 实现*
- *线性神经网络及 MATLAB 实现*
- *径向基函数网络及 MATLAB 实现*
- *GMDH 网络及 MATLAB 实现*

3.1　感知器网络及 MATLAB 实现

感知器网络是由 M-P 网络发展而来的，它是一个线性阈值单元组成的网络，它的结构和学习算法非常简单，是其他前向型神经网络的基础。特别地，由于感知器的隐含层神经元是线性的，因此，隐单元的选取相对于其他由非线性阈值单元组成的网络（如 BP 网络等）要容易得多。对感知器网络的研究、分析与设计，可以为其他网络的分析与设计提供理论依据和知识基础。

感知器网络分为单层感知器和多层感知器神经网络。本节针对这两类感知器展开讨论。如果没有特别注明，以后都用感知器作为单层感知器的简称。

3.1.1　单层感知器网络

1. 单层感知器网络理论

感知器是在 M-P 模型的基础上建立起来的单神经元的网络，其网络基本结构如图 3-1 所示。网络的基本特性，也就是神经元的工作特性，可归纳为以下几点。

（1）感知器是一个多输入、单输出的运算系统，表示了神经元的运算特性。它的输入状态向量可记为：

$$x_n=(x_1,x_2,\cdots,x_n)$$

其中，输入分量 x_i 表示第 i 个神经元的状态。

记 $w_n=(w_1, w_2,\cdots, w_n)$为权向量，其中 w_i 表示第 i 个神经元与感知器的连接权重。

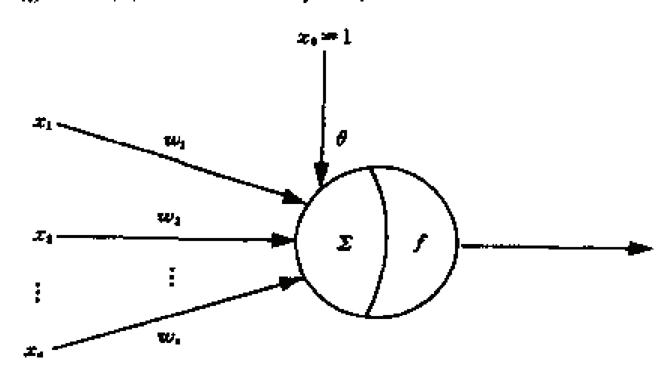


图 3-1　感知器网络基本结构

（2）感知器的状态值可以作为感知器的输出，它是由输入向量 x_n 、权值向量 ω_n 和阈值向量 h 共同决定的，通过一个运算函数 $T(\cdot)$ 得到输出。最常用的运算函数为：

$$z=f\left(\sum_{i=1}^{n}\omega_i x_i-h\right)=f\left(\sum_{i=0}^{n}\omega_i x_i\right)$$

其中，$\omega_0=-h$（阈值向量）$f(u)$，为位于区间[－1,1]之间的单调递增函数，称为激励函数。而

$$u=\sum_{i=1}^{n}\omega_i x_i-h$$

称为感知器的整合函数。激励函数的要求是连续可微、单调递增和对称的，例如，可将激励函数定义为：

$$f(u)=\frac{2}{1+\exp(-u)}-1$$

感知器的基本功能是对外部的神经元进行“感知”与“识别”。也就是说，当外部的神经元处于一定的状态时，感知器就呈现“兴奋”状态；而当外部的神经元处于另一种状态时，感知器就呈现“抑制”状态。因此，感知器的输出一般为 0 或 1 状态，这样就可以很容易地对模式进行分类。

感知器的学习是有教师的学习，权值可以通过学习进行调整。学习算法的步骤如下。

（1）设置权重的初始值 $\omega_i(0)$（$j=0,1,\cdots n$）为较小的随机非 0 值。



（2）给定输入、输出样本向量，也就是导师信号 $x_p/d_p, p=1,2,\cdots,L$，其中，

$$x_p=\left(x_{0p},x_{1p},\cdots,x_{np}\right)$$

$$d_p=\begin{cases}1, x_p\in A\\-1, x_p\in B\end{cases}$$

式中，$x_{0p}=1$。

（3）求感知器的输出 $z_p(t)$

$$z_p(t)=f\left(\sum_{j=0}^{n}\omega_j(t)x_{jp}\right)$$

（4）权值调整：

$$\omega_j(t+1)=\omega_j(t)+\eta\left(d_p-z_p(t)\right)x_{jp}$$

其中，t 表示第 t 次调整权值；η 表示学习速率，用于控制权值调整速度，且$0<\eta\leqslant1$。

（5）如果 $z_p(t)=d_p$，则学习停止；否则，返回第（3）步。

感知器的算法规则属于梯度下降法，如果目标向量存在，那么经过有限次循环迭代后，一定能够收敛到正确的目标向量。由此可见，学习结束后的网络，将样本模式以权值和阈值的形式，分别记忆（存储）于网络中。

另外，对于单层感知器，有一个重要的定理，即如果输入的两类模式是线性可分集合，那么学习算法一定收敛。也就是说，感知器一定可以成功地将两类模式进行正确分类。

由此，可以得出另外一个结论，即如果输入的两类模式并非线性可分的，那么学习算法就不收敛。也就是说，感知器只可以对线性可分的向量集合进行分类。而判断一组向量集合是否为线性可分是相当困难的，尤其是当向量集合的元素数量非常大时。因此，如果尝试利用感知器对非线性可分的向量集合进行分类时，学习算法就会处于一种无休止的循环状态，浪费大量的计算资源，这是感知器存在的一个比较严重的缺陷。

此外，如果输入样本向量集合中存在奇异样本（即该样本相对于其他样本特别的大或特别的小），学习算法就会运算得很慢。

最后，由于感知器的输出只有 0 和 1 两种状态，因此它只能对输入模式进行比较简单的分类。

2. 单层感知器的 MATLAB 实现

MATLAB 7 的神经网络工具箱中为单层感知器的设计、训练和学习等提供了专门的函数工具，包括：

- 设计函数 newp
- 训练函数 learnp 和 learnpn

- 自适应函数 adapt 和 adaptwb
- 训练函数 train 和 trainwb
- 网络仿真函数 sim
- 初始化函数 init
- 传递函数 hardlim 和 hardlims

可以利用以上函数进行感知器的设计、分析及应用，这些函数的功能及调用格式等已经在第 2 章中进行了详细说明。前面已经提到，感知器的主要功能就是进行模式分类，在给定了输入向量 P、目标向量 T 后，可以通过某种学习规则对输入模式进行成功分类。

下面，我们以设计一个双输入的单层感知器为例，详细介绍如何设计并应用一个单层感知器的方法。该感知器的网络结构如图 3-2 所示。由图可见，其采用的传递函数为 hardlim。

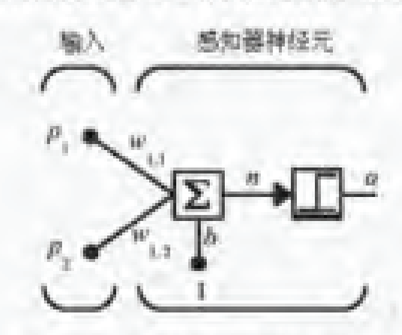


图 3-2　单层感知器的网络结构

首先，在利用函数 newp 设计一个感知器网络之前，需要先设定输入向量的范围向量 PT，PT 是一个 $R\times2$ 维的矩阵，其中 R 为输入向量 P 的维数。

```
PT=[0 2, -2 2];
net=newp(PT,1,'','')
```

其中，参数“1”表示只有一个神经元，由 newp 的调用格式可知，后面的两个参数为''，表示采用函数的默认值：分别设定感知器的传递函数为 harlim、学习函数为 learnp。因此，net 是满足要求的单层感知器。如果设计一个传递函数为 hardlims、学习函数为 learnpn 的单层感知器，newp 的调用格式为：

```
net=newp(PT,1,'hardlims','learnpn')
```

设计好的感知器并不能马上投入使用。通过样本训练，确定感知器的权值和阈值。感知器网络的训练过程为：对于给定的输入向量，计算网络的实际输出，并与相应的目标向量进行比较，得到误差 ε，然后根据相应的学习规则调整权值和阈值。重新计算网络在新的权值和阈值作用下的输出，重复上述的权值和阈值的调整过程，直到网络的输出与期望的目标向量相等或者训练次数达到预定的最大次数时才停止训练。之所以要设定最大训练次数，是因为对于有些问题，感知器的学习算法可能不会收敛，这也是感知器网络的严重缺陷之一。这里利用函数 train 对感知器进行训练，调用格式为：

```
net.trainParam.epochs = 10;
net=train(new,P,T)
```

代码中的第一行表示预先设定的最大训练次数为 10 次，感知器经过 10 次训练后停止。如果在 10 次以内，感知器已经得到了理想的输出，则训练自动停止。

感知器的输出是通过仿真函数 sim 得到的，调用格式为：

```
Y=sim(net,P)
```

Y 表示感知器的输出向量，该函数在一定的权值和阈值下，根据给定的输入向量，确定感知器的输出。

例 3.1 给定样本输入向量 P、目标向量 T 及需要进行分类的输入向量组 Q，设计一个单层感知器，对其进行分类。

MATLAB 代码如下：

```
P=[-0.5 -0.6 0.7; 0.8 0 0.1];
T=[1 1 0];
net=newp([-1 1;-1 1],1);
%返回画线的句柄，下一次绘制分类线时将旧的删除
handle=plotpc(net.iw{1},net.b{1});
%设置训练次数最大为 10 次
net.trainParam.epochs = 10;
net=train(net,P,T);
%给定的输入向量
Q=[0.6 0.9 -0.1; -0.1 -0.5 0.5];
Y=sim(net,Q);
figure;
%绘制分类线
plotpv(Q,Y);
handle=plotpc(net.iw{1},net.b{1},handle)
```

运行结果如图 3-3 所示。由图可见，所设计的感知器对输入模式进行了成功的分类。

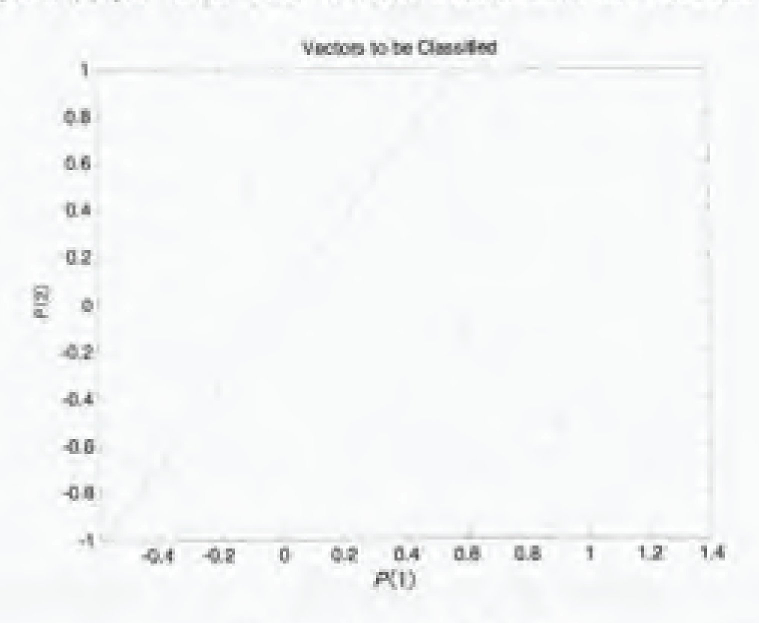


图 3-3　输入向量及分类线

感知器训练结果为：

```
TRAINC, Epoch 0/10
TRAINC, Epoch 3/10
TRAINC, Performance goal met.
```

可见，经过 3 次训练后，网络目标误差达到要求，如图 3-4 所示。

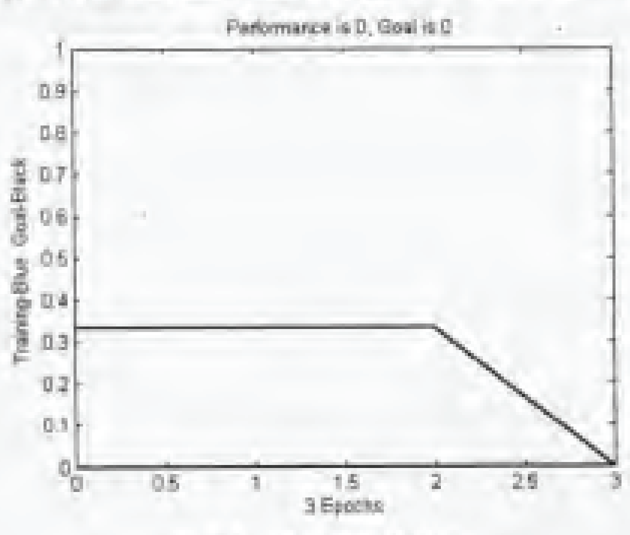


图 3-4　感知器训练过程

下面给出一个线性不可分的例子。给定一个双元素的输入向量组 P，每列都由 5 个元素组成，并给定一个目标向量 T。利用以下代码可将其绘制在一个平面上。运行结果如图 3-5 所示。

```
P = [ -0.5 -0.5 +0.3 -0.1 -0.8; ...
      -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(P,T)
```

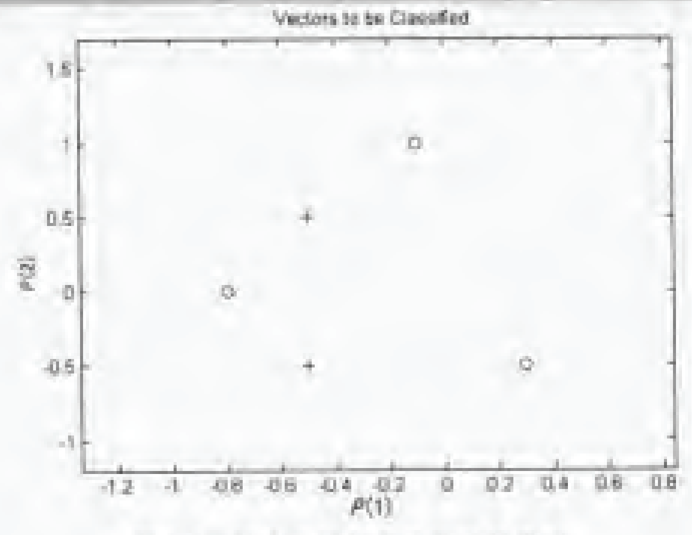


图 3-5　样本点的分布及相应的类别

接下来尝试设计一个感知器，该感知器必须将输入向量 P 进行准确的分类。利用函数 newp 创建一个感知器。

```
net = newp([-40 1;-1 50],1);
```

在利用感知器进行分类之前，首先需要对感知器进行初始化，将其权值设定为 0。这样一来，任何输入都将产生同样的输出，感知器也就不能进行分类了。

感知器必须经过训练才能应用，在这里使用自适应函数 adapt 对其进行训练，自适应的循环次数设定为 3 次，经过 25 次迭代后，训练停止。代码如下：

```
net.adaptParam.passes = 3;
```

```
linehandle=plotpc(net.IW{1},net.b{1});
for a = 1:25
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);   drawnow;
end;
```

对输入样本的分类结果如图 3-6 所示。由图可见，对于这种线性不可分的模式，利用单层感知器是无法进行正确分类的。

该实例的完整 MATLAB 代码如下：

```
P = [ -0.5 -0.5 +0.3 -0.1 -0.8; ...
      -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(P,T);
net = newp([-40 1;-1 50],1);
%对训练前的样本进行绘图，并获得绘图的句柄 linehandle
hold on
plotpv(P,T);
linehandle=plotpc(net.IW{1},net.b{1});
%设定自适应循环次数为 3
net.adaptParam.passes = 3;
linehandle=plotpc(net.IW{1},net.b{1});
for a = 1:25
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);   drawnow;
end;
```

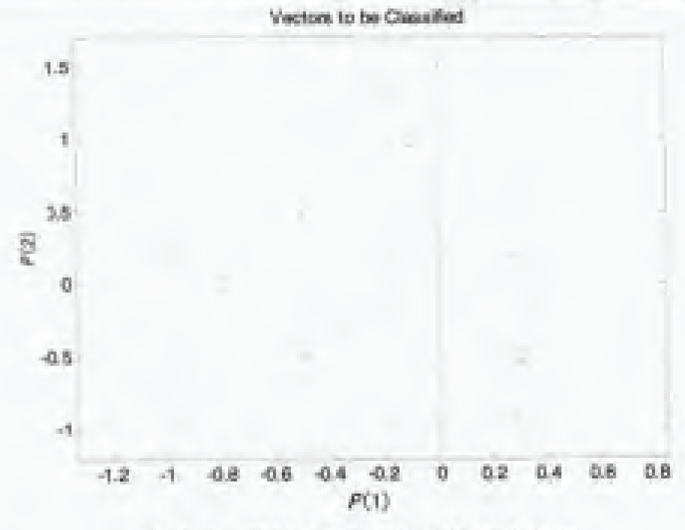


图 3-6　线性不可分样本的分类结果

3.1.2　多层感知器

1. 多层感知器理论

多层感知器是单层感知器的一种推广形式。图 3-7 给出了一个三层感知器的网络结构，

由图可见，输入结点为 n 个，即 $x_1,x_2,\cdots,x_n$；第一隐层有 n_1 个神经元，对应的输出有 $h_1,h_2,\cdots,h_{n_1}$；第二隐层有 n_2 个神经元，对应的输出为 $c_1,c_2,\cdots,c_{n_2}$；整个网络的输出为 y。整个网络按照前馈的方式进行连接，输入输出关系为：

$$\begin{cases} h_j = f\left(\sum_{i=1}^{n} w_{ij}x_i - \theta_j\right) & j=1,2,\cdots,n_2 \\ c_k = f\left(\sum_{i=1}^{n_1} v_{jk}h_j - \theta_k\right) & k=1,2,\cdots,n_1 \\ y = f\left(\sum_{k=1}^{n_1} w_k c_k - \theta\right) \end{cases}$$

上式中，h_j 为第一隐层第 j 个单元的输出，w_{ij} 为输入层第 i 个结点与第一隐层第 j 个单元之间的连接权值，θ_j 为第一隐层第 j 个单元的阈值；同理，c_k 为第二隐层第 k 个单元的输出，θ_k 为其对应的阈值，v_{jk} 是第一隐层第 j 个单元与第二隐层第 k 个单元间的连接权值，y 为网络输出，w_k 为第二隐层第 k 个单元与输出之间的连接权。式中的传递函数 $f(\cdot)$ 为 0-1 函数，这也是每一个单层感知器的传递函数。

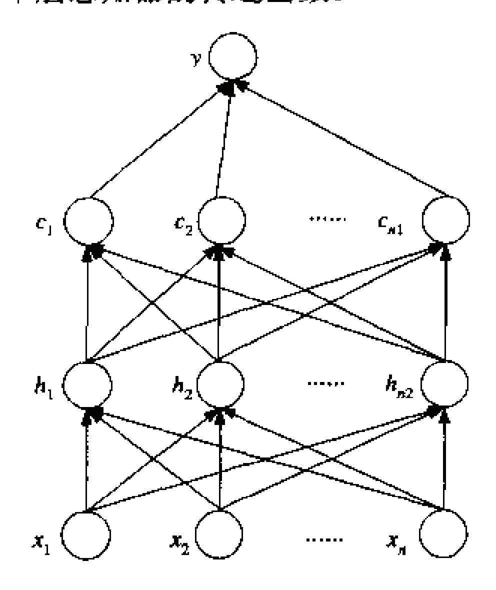


图 3-7　多层感知器网络结构

多层感知器网络的信息是逐层前向传播的，下层的各单元与上一层的每个单元相连。输入单元按照输入/输出关系式逐层进行操作，每层之间的连接权值可以通过学习规则进行调整。可以看出，多层感知器实际上就是多个单层感知器经过适当组合设计而成的，它可以实现任何形状的划分。

多层感知器的隐单元的数目 n_1 有对应的上下限公式，可供设计时参考。如果需要对 k 个输入样本进行分类，隐单元数目的最大值为 $k-1$，最小值为 $n_1=\min\left[p\left(n_1,n\right)\right]\geqslant k$，其中，$p\left(n_1,n\right)=\sum_{i=0}^{n}\binom{n_1}{i}$，当 $n_1<i$ 时，$\binom{n_1}{i}=0$。

下面通过设计一个二层感知器来解决“异或”问题，进而说明多层感知器的设计过程及功能。异或问题的输入样本 P 和目标向量 T 如表 3-1 所示。从几何意义上看，输入向量相当于二维平面上一个正方形的 4 个顶点。由于不存在一条直线可以将两个顶点（0,1）、（1,0）与另外两个顶点（0,0）、（1,1）分开，所以单层感知器是无法实现异或功能的。

感知器的网络结构按此方案确定：输入结点 n=2，输出结点 m=1。隐层单元数目 n_1 由这两个公式确定，可以看出 $2\leqslant n_1\leqslant 3$，这里我们取 $n_1=2$。在多层感知器的结构确定以后，可以利用一定的学习规则对网络进行训练。我们所设计的感知器的网络结构如图 3-8 所示。

表 3-1　异或问题输入及输出关系

输入样本 P	目标向量 T
0　0	0
0　1	1
1　0	1
1　1	0

感知器的训练由一组样本组成的集合进行。在训练期间，将这些样本重复送到感知器的输入层，通过调整权值和阈值使感知器的输出达到目标输出。

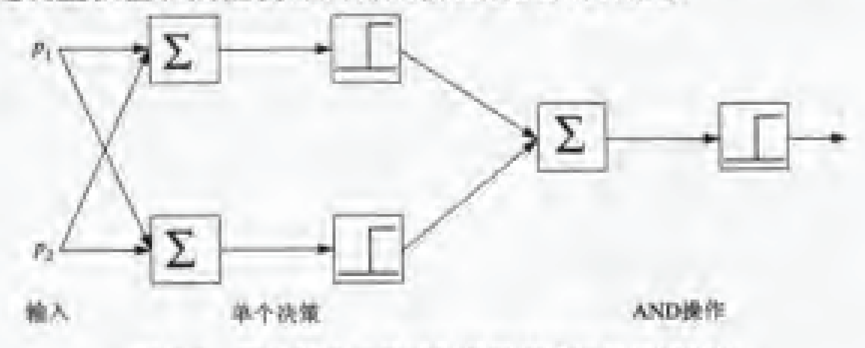


图 3-8　用于解决“异或”问题的感知器网络结构

多层感知器可对输入空间向量进行分类，它的输入/输出之间的映射在一定程度上是确定的。一方面，它的抗干扰能力和鲁棒性比较差，在样本存在奇异值或噪声的情况下，不能得到正确的分类；另一方面，为了实现非线性划分，需要增加感知器的层数，这就增加了计算难度，使得感知器的应用范围受到了限制。

由于神经网络工具箱中没有为多层感知器提供专门的函数，似乎是没有办法利用 MATLAB 实现多层感知器了，其实不然，MATLAB 的神经网络工具箱中的定制网络功能

可以帮助我们解决这个问题。接下来以图 3-8 所示的多层感知器为例，详细介绍如何在神经网络工具箱中定制网络。

2. 网络定义

首先生成一个新的网络，在命令行窗口中键入以下代码后就可以生成一个类型未知的网络。该网络不仅类型未知，并且包括层数、输入/输出量和权值等都是未知的。在命令行中输入 net 后按回车键，就会得到网络的属性。我们发现，目前所有的属性都是零值或者为空。

```
net=network
```

接下来，需要设置网络的结构属性。这些属性包括网络的输入层的数目、输入层之间的连接情况。本例的设置如下：

```
net.numInputs=1;
net.numLayers=2;
```

这意味着网络是一个两层网络，有一个输入源。



net.numInputs 指的是输入源的数目，而不是在一个输入向量中的元素的数目。元素的数目即网络输入层神经元的个数，设置格式为 net.layers{1}.size。

然后设置网络的阈值、输入权值、层的权值、输出与目标的连接情况。本例中，网络本身是一个前馈网络，所以层连接应该设置为[0 0;1 0]，即信息只有通过第 1 层才可以传播到第 2 层，网络的输入只和第 1 层相连，两层都存在权值向量。第 2 层存在目标向量，输出也处于该层。因此，属性的设置代码为：

```
net.biasConnect=[1;1];
net.inputConnect=[1;0];
net.layerConnect=[0 0;1 0];
net.targetConnect=[0 1];
net.outputConnect=[0 1];
```

接下来，需要设置各层的属性，包括神经元的个数、传递函数、输入向量的范围和初始化函数等。本例中，多层感知器的第 1 层有 2 个神经元，传递函数为硬限幅函数 hardlim，输入向量的范围为[0 1]；输出层即第 2 层，有 1 个神经元，传递函数亦为硬限幅函数 hardlim。两层的初始化函数都采用 Nguyen-Wodrow 函数 initnw。设置格式为：

```
net.inputs{1}.range=[0 1;0 1];
net.layers{1}.size=2;
net.layers{1}.transferFcn='hardlim';
net.layers{1}.initFcn='initnw';
net.layers{2}.size=1;
net.layers{2}.transferFcn='hardlim';
net.layers{2}.initFcn='initnw';
```

最后，需要设置网络的整体属性，即性能函数、训练函数和初始化函数等，本例中，性能函数区均方差函数为 mse，训练函数为 trainlm，初始化函数为 initlay。设置格式为：

```
net.adaptFcn='trains';
net.performFcn='mse';
net.trainFcn='trainlm';
```

```
net.initFcn='initlay';
```

到此为止，一个完整的多层感知器就设计好了。下一步就是设置训练参数对其进行训练，利用其实现异或功能。训练代码为：

```
%对网络进行初始化
net=init(net);
%设置训练样本P和T
P=[0 0;0 1;1 0;1 1]';
T=[0 1 1 0];
%设置网络训练次数500
net.trainParam.epochs=500;
net=train(net,P,T);
```

网络训练结果为：

```
TRAINLM, Epoch 0/500, MSE 0.46261/0, Gradient 0.242252/1e-010
TRAINLM, Epoch 17/500, MSE 6.50732e-012/0, Gradient 4.49737e-011/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

可见，利用trainlm函数进行训练，在第18次的时候，达到了最小梯度，但网络目标误差0没有达到，此时的训练误差为6.50732e-012，如图3-9所示。

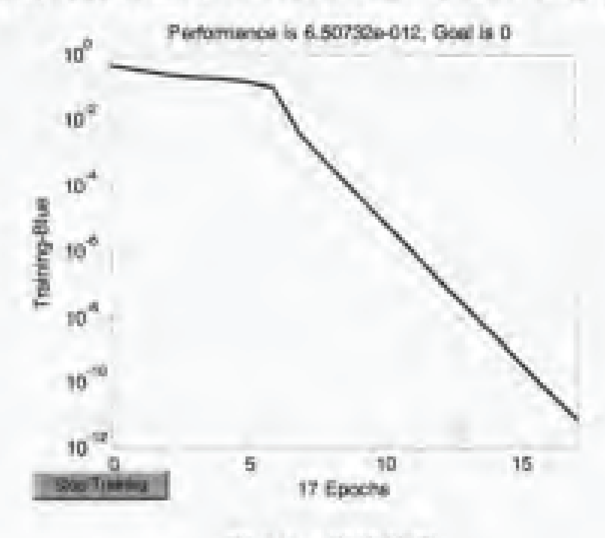


图3-9 训练结果

为了检验网络的性能是否达到要求，在命令行窗口中输入：

```
y=sim(net,P)
```

运行结果为：

```
0.0000    1.0000    1.0000    0.0000
```

可见，此时的网络已经实现了异或功能，这是因为6.50732e-012的误差实际上已经非常小了。

3.2 BP网络及MATLAB实现

BP网络是一种多层前馈神经网络，名字源于网络权值的调整规则采用的是后向传播学习算法，即BP学习算法。BP学习算法是Rumelhart等在1986年提出的。自此以后，BP

神经网络获得了广泛的实际应用。据统计，80%～90%的神经网络模型采用了 BP 网络或者它的变化形式。BP 网络是前向网络的核心部分，体现了神经网络中最精华、最完美的内容。

3.2.1 BP 网络理论

1. BP 网络结构

BP 网络是一种单向传播的多层前向网络，其结构如图 3-10 所示。由图可见，BP 网络是一种具有三层或三层以上的神经网络，包括输入层、中间层（隐层）和输出层。上下层之间实现全连接，而每层神经元之间无连接。当一对学习样本提供给网络后，神经元的激活值从输入层经各中间层向输出层传播，在输出层的各神经元获得网络的输入响应。接下来，按照减少目标输出与实际误差的方向，从输出层经过各中间层逐层修正各连接权值，最后回到输入层，这种算法称为“误差逆传播算法”，即 BP 算法。随着这种误差逆的传播修正不断进行，网络对输入模式响应的正确率也不断上升。

与感知器模型不同的是，BP 网络的传递函数要求必须是可微的，所以不能使用感知器网络中的二值函数，常用的有 Sigmoid 型的对数、正切函数或线性函数。由于传递函数是处处可微的，所以对于 BP 网络来说，一方面，所划分的区域不再是一个线性划分，而是由一个非线性超平面组成的区域，它是比较平滑的曲面，因而它的分类比线性划分更加精确，容错性也比线性划分更好；另一方面，网络可以严格采用梯度下降法进行学习，权值修正的解析式十分明确。

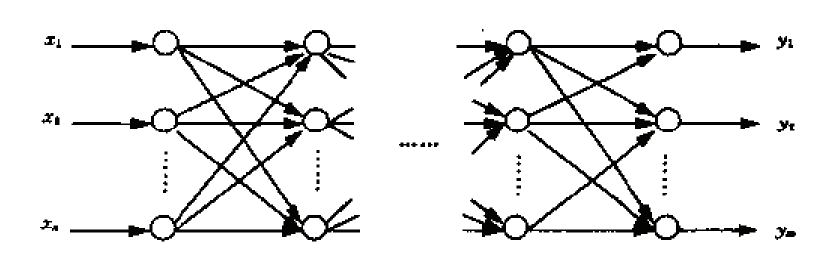


图 3-10 BP 网络结构

2. BP 网络学习规则

这里不讨论学习规则的数学推导过程，只给出学习过程及步骤，以供设计与分析 BP 网络时作为参考。我们以一个三层 BP 网络为例，介绍 BP 网络的学习过程及步骤。

在介绍之前，首先对各符号的形式及意义进行说明。

网络输入向量 $P_k=(a_1,a_2,\cdots,a_n)$；

网络目标向量 $T_k=(y_1,y_2,\cdots,y_q)$；

中间层单元输入向量 $S_k=(s_1,s_2,\cdots,s_p)$，输出向量 $B_k=(b_1,b_2,\cdots,b_p)$；

输出层单元输入向量 $L_k=(l_1,l_2,\cdots,l_q)$，输出向量 $C_k=(c_1,c_2,\cdots,c_q)$；

输入层至中间层的连接权 w_{ij}，$i=1,2,\cdots,n$，$j=1,2,\cdots,p$；
中间层至输出层的连接权 v_{jt}，$j=1,2,\cdots,p$，$t=1,2,\cdots,p$；
中间层各单元的输出阈值 θ_j，$j=1,2,\cdots,p$；
输出层各单元的输出阈值 γ_j，$j=1,2,\cdots,p$；
参数 $k=1,2,\cdots$，m。

（1）初始化。给每个连接权值 w_{ij}、v_{jt}、阈值 θ_j 与 γ_t 赋予区间（－1,1）内的随机值。

（2）随机选取一组输入和目标样本 $P_k=\left(a_1^k,a_2^k,\cdots,a_n^k\right)$、$T_k=\left(s_1^k,s_2^k,\cdots,s_p^k\right)$ 提供给网络。

（3）用输入样本 $P_k=\left(a_1^k,a_2^k,\cdots,a_n^k\right)$、连接权 w_{ij} 和阈值 θ_j 计算中间层各单元的输入 s_j，然后用 s_j 通过传递函数计算中间层各单元的输出 b_j。

$$s_j=\sum_{i=1}^{n}w_{ij}a_i-\theta_j \qquad j=1,2,\cdots,p$$

$$b_j=f\left(s_j\right) \quad j=1,2,\cdots,p$$

（4）利用中间层的输出 b_j、连接权 v_{jt} 和阈值 γ_t 计算输出层各单元的输出 L_t，然后利用通过传递函数计算输出层各单元的响应 C_t。

$$L_t=\sum_{j=1}^{p}v_{jt}b_j-\gamma_t \quad t=1,2,\cdots,q$$

$$C_t=f\left(L_t\right) \quad t=1,2,\cdots,q$$

（5）利用网络目标向量 $T_k=\left(y_1^k,y_2^k,\cdots,y_q^k\right)$，网络的实际输出 C_t，计算输出层的各单元一般化误差 d_t^k。

$$d_t^k=\left(y_t^k-C_t\right)\cdot C_t\left(1-C_t\right) \quad t=1,2,\cdots,q$$

（6）利用连接权 v_{jt}、输出层的一般化误差 d_t 和中间层的输出 b_j 计算中间层各单元的一般化误差 e_j^k。

$$e_j^k=\left[\sum_{t=1}^{q}d_t\cdot v_{jt}\right]b_j\left(1-b_j\right)$$

（7）利用输出层各单元的一般化误差d_t^k与中间层各单元的输出b_j来修正连接权v_{jt}和阈值γ_t。

$$v_{jt}\left(N+1\right)=v_{jt}\left(N\right)+\alpha\cdot d_t^k\cdot b_j$$
$$\gamma_t\left(N+1\right)=\gamma_t\left(N\right)+\alpha\cdot d_t^k$$
$$t=1,2,\cdots,q,\ j=1,2,\cdots,p,\ 0<\alpha<1$$

（8）利用中间层各单元的一般化误差e_j^k，输入层各单元的输入$P_k=\left(a_1,a_2,\cdots,a_n\right)$来修正连接权$w_{ij}$和阈值$\theta_j$。

$$w_{ij}\left(N+1\right)=w_{ij}\left(N\right)+\beta e_j^k a_i^k$$
$$\theta_j\left(N+1\right)=\theta_j\left(N\right)+\beta e_j^k$$
$$i=1,2,\cdots,n,\ j=1,2,\cdots,p,\ 0<\beta<1$$

（9）随机选取下一个学习样本向量提供给网络，返回到步骤（3），直到 m 个训练样本训练完毕。

（10）重新从 m 个学习样本中随机选取一组输入和目标样本，返回步骤（3），直到网络全局误差 E 小于预先设定的一个极小值，即网络收敛。如果学习次数大于预先设定的值，网络就无法收敛。

（11）学习结束。

可以看出，在以上学习步骤中，（7）～（8）步为网络误差的“逆传播过程”，（9）～（10）步则用于完成训练和收敛过程。

通常，经过训练的网络还应该进行性能测试。测试的方法就是选择测试样本向量，将其提供给网络，检验网络对其分类的正确性。测试样本向量中应包含今后网络应用过程中可能遇到的主要典型模式。这些，样本可以通过直接测取得到，也可以通过仿真得到，在样本数据较少或者较难得到时，也可以通过对学习样本加上适当的噪声或按照一定规则插值得到。总之，一个良好的测试样本集中不应该包含和学习样本完全相同的模式。

3. BP 网络设计技巧

（1）输入和输出层的设计

输入的神经元可以根据需要求解的问题和数据表示方式确定。如果输入的是模拟信号波形，那么输入层可以根据波形的采样点数目决定输入单元的维数，也可以用一个单元输入，这时输入样本为采样的时间序列；如果输入为图像，则输入单元可以为图像的像素，也可以是经过处理的图像特征。

输出层的维数可根据使用者的要求确定。如果将 BP 网络用做分类器，类别模式一共有 m 个，那么输出层神经元的个数为 m 或 $\log_2 m$ 。

（2）隐层的设计

对于 BP 网络，有一个非常重要的定理。即对于任何在闭区间内的一个连续函数都可以用单隐层的 BP 网络逼近，因而一个三层 BP 网络就可以完成任意的 n 维到 m 维的映射。

隐层的神经元数目选择是一个十分复杂的问题，往往需要根据设计者的经验和多次实验来确定，因而不存在一个理想的解析式来表示。隐单元的数目与问题的要求、输入/输出单元的数目都有着直接关系。隐单元数目太多会导致学习时间过长、误差不一定最佳，也会导致容错性差、不能识别以前没有看到的样本，因此一定存在一个最佳的隐单元数。以下 3 个公式可用于选择最佳隐单元数时的参考公式。

1）$\sum_{i=0}^{n} C_{n_1}^{i} > k$，其中，$k$ 为样本数，n_1 为隐单元数，n 为输入单元数。如果 $i > n_1$，$C_{n_1}^{i} = 0$。

2）$n_1 = \sqrt{n+m} + a$，其中，m 为输出神经元数，n 为输入单元数，a 为[1,10]之间的常数。

3）$n_1 = \log_2 n$，其中，n 为输入单元数。

还有一种途径可用于确定隐单元的数目。首先使隐单元的数目可变，或者放入足够多的隐单元，通过学习将那些不起作用的隐单元剔除，直到不可收缩为止。同样，也可以在开始时放入比较少的神经元，学习到一定次数后，如果不成功则再增加隐单元的数目，直到达到比较合理的隐单元数目为止。

4. 初始值的选取

由于系统是非线性的，初始值对于学习能否达到局部最小和是否能够收敛的结果关系很大。一个重要的要求是：初始权值在输入累加时使每个神经元的状态值接近于零，权值一般取随机数，要比较小。输入样本也同样希望进行归一化处理，使那些比较大的输入仍落在传递函数梯度大的地方。

5. BP 网络的不足及改进

虽然 BP 网络得到了广泛应用，但其自身也存在一些缺陷和不足，主要包括几个方面的问题。

首先，由于学习速率是固定的，因此，网络的收敛速度慢，需要较长的训练时间。对于一些复杂的问题，BP 算法需要的训练时间可能会非常长。这主要是由于学习速率太小造成的，可采用变化的学习速率或自适应的学习速率加以改进。

其次，BP 算法可以使权值收敛到某个值，但并不能保证其为误差平面的全局最小值，这是因为采用梯度下降法可能会产生一个局部最小值。对于这个问题，可以采用附加动量法来解决。

再次，网络隐含层的层数和单元数的选择尚无理论上的指导，一般是根据经验或者通过反复实验确定。因此，网络往往存在很大的冗余性，在一定程度上也增加了网络学习的负担。

最后，网络的学习和记忆具有不稳定性。也就是说，如果增加了学习样本，训练好的网络就需要从头开始重新训练，对于以前的权值和阈值是没有记忆的。

3.2.2 BP网络的MATLAB设计

下面以一个单隐层的BP网络设计为例，介绍利用神经网络工具箱进行BP网络设计及分析的过程。

前面已经提到，对于BP网络，存在一个重要的结论，即单隐层的BP网络可以逼近任意的非线性映射，前提是隐含层的神经元个数可以随意调整。因此，为了简单起见，这里利用一个单隐层的BP网络来逼近一个函数。

1. 问题描述

通过对函数进行采样得到了网络的输入变量P和目标变量T，分别为：

```
P =-1:0.1:1
T= [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600 0.4609 0.1336 -0.2013 -0.4344 -0.5000 -0.3930
-0.1647 0.0988 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
```

每组向量都有21组数据，可以将输入向量和目标向量绘制在一起，如图3-11所示。

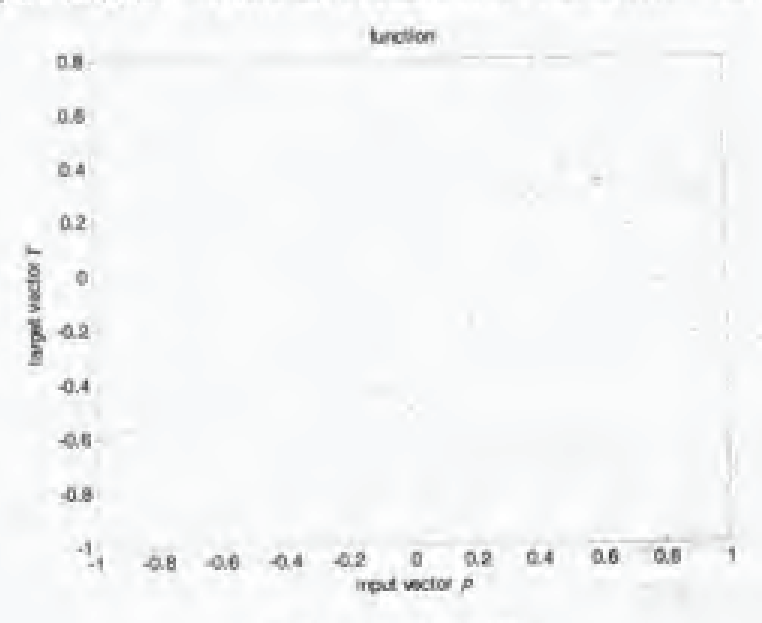


图3-11 函数原型

2. 网络设计

该网络的输入层和输出层的神经元个数均为1，根据以上的隐含层设计经验公式，以及考虑本例的实际情况，解决该问题的网络的隐层神经元个数应该在3～8之间。因此，下面设计一个隐含层神经元数目可变的BP网络，通过误差对比，确定最佳的隐含层神经元个数，并检验隐含层神经元个数对网络性能的影响。

网络的设计及训练代码如下：

```
s=3:8;
res=1:6;
for i=1:6
```

```
        net=newff(minmax(P),[s(i),1],{'tansig','logsig'},'traingdx');
        net.trainParam.epochs=2000;
        net.trainParam.goal=0.001;
        net=train(net,P,T)
        y=sim(net,P);
        error=y-T;
        res(i)=norm(error);
    end
```

由此可见，网络的隐含层神经元的传递函数为 tansig，输出层神经元的传递函数为 logsig，这是因为目标向量的元素都位于区间[－1,1]中，正好满足函数 tansig 的输出要求。上述代码的运行结果见表 3-2。

表 3-2　网络训练误差

神经元个数	3	4	5	6	7	8	9	10
网络误差	1.0412	0.7297	0.1767	0.1449	0.1807	0.1442	0.1449	0.1621

表 3-2 表明，在经过 2000 次训练后（训练函数采用 traingdx），隐含层神经元为 8 的 BP 网络对函数的逼近效果最好，因为它的误差最小，而且网络经过 232 次训练就达到了目标误差。隐含层为 6 和 9 的网络误差也比较小，但它们所需要的训练时间比较长。考虑到网络性能的训练速度，这里将网络隐含层的神经元数目设定为 8。



从表 3-2 中还可得出一个重要的结论，就是并非隐含层神经元的个数越多，网络的性能就越好。在本例中，误差并没有明显地随着隐含层神经元数目的增加而减小的趋势，如当隐含层神经元的个数从 6 增加到 7 时，误差反而增大了，从 8 增加到 9 和从 9 增加到 10 也观察到了这个现象。

采用不同的训练函数对网络的性能也有影响，比如收敛速度等。下面采用不同的训练函数对网络进行训练，并观察结果。

以上都是采用函数 traingdx 对网络进行训练，该函数的学习算法是梯度下降动量法，而且学习速率是自适应的。当隐含层神经元数目为 8 时，网络的逼近误差为 0.1442。网络的训练结果如图 3-12 所示。

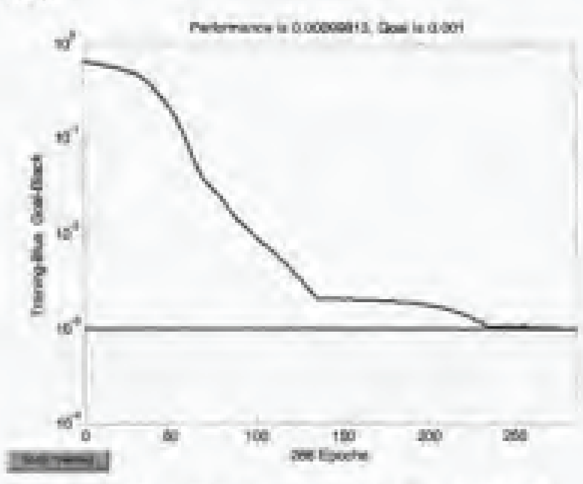


图 3-12　训练结果（训练函数：traingdx）

接下来采用函数 trainlm 对网络进行训练，该函数的学习算法为 Levenberg-Marquadt 反传算法，该训练函数的优点在于收敛速度很快。训练代码为：

```
net.trainParam.epochs=2000;
net.trainParam.goal=0.001;
net=train(net,P,T);
y=sim(net,P);
error=y-T;
res=norm(error);
```

训练结果为：

```
TRAINLM, Epoch 0/2000, MSE 0.44874/0.001, Gradient 3.73574/1e-010
TRAINLM, Epoch 5/2000, MSE 0.000993055/0.001, Gradient 0.31458/1e-010
TRAINLM, Performance goal met.
```

可见，经过 5 次训练后，网络的目标误差就达到了要求。如图 3-13 所示，此时的误差 res=0.1444。

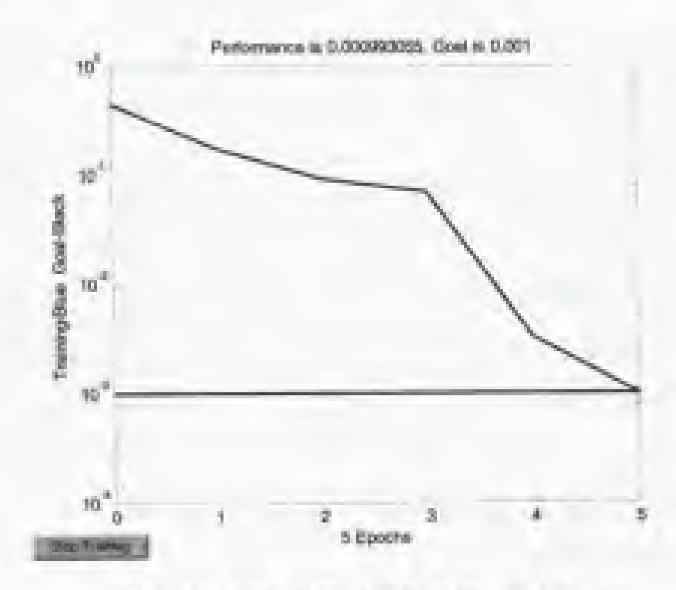


图 3-13　训练结果（训练函数：trainlm）

最后采用 traingd 对网络进行训练，该函数所用的学习算法就是普通的梯度下降法。训练代码为：

```
net.trainParam.epochs=2500;
net.trainParam.goal=0.001;
net=train(net,P,T);
y=sim(net,P);
error=y-T;
res=norm(error);
```

训练结果为：

```
TRAINGD, Epoch 2500/2500, MSE 0.00818525/0.001, Gradient 0.0151572/1e-010
TRAINGD, Maximum epoch reached, performance goal was not met.
```

可见，经过 2500 次训练后，网络目标误差目标依然没有被满足，而且通过观察网络的训练曲线，如图 3-14 所示，网络的训练过程收敛得非常缓慢，而且训练误差 res=0.4146。

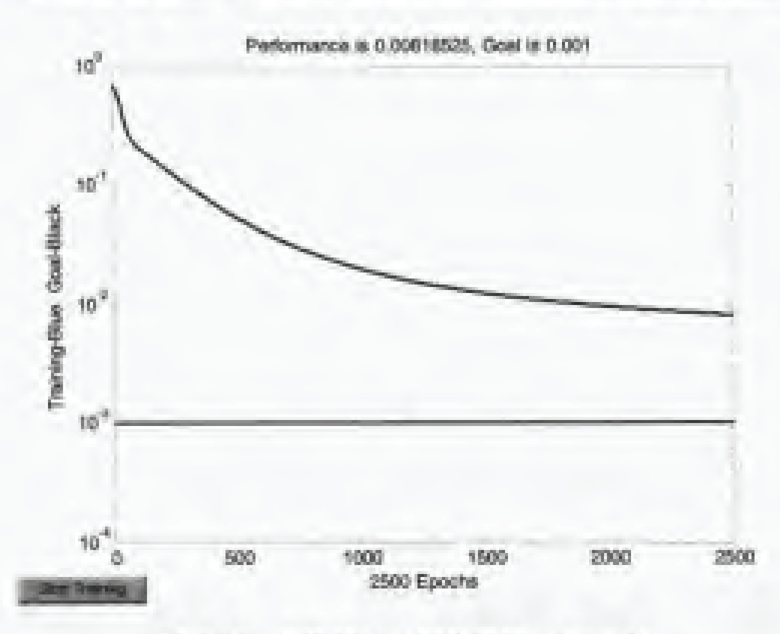


图 3-14 训练结果（训练函数：traingd）

综合分析以上 3 种训练过程，我们认为，函数 trainlm 收敛速度快，网络的训练误差也比较小。因此，这里采用 trainlm 对网络进行训练。通过对训练好的网络进行仿真，可以得到网络对函数的逼近情况，代码如下：

```
y=sim(net,P);
plot(P,T,'r+');
hold on
plot(P,y,'.');
```

运行结果如图 3-15 所示，其中，黑点表示网络的输出结果，"+" 表示函数的实际值。利用以下代码可以绘制网络的误差曲线：

```
plot(1:21,y-T);
```

网络的误差曲线如图 3-16 所示。

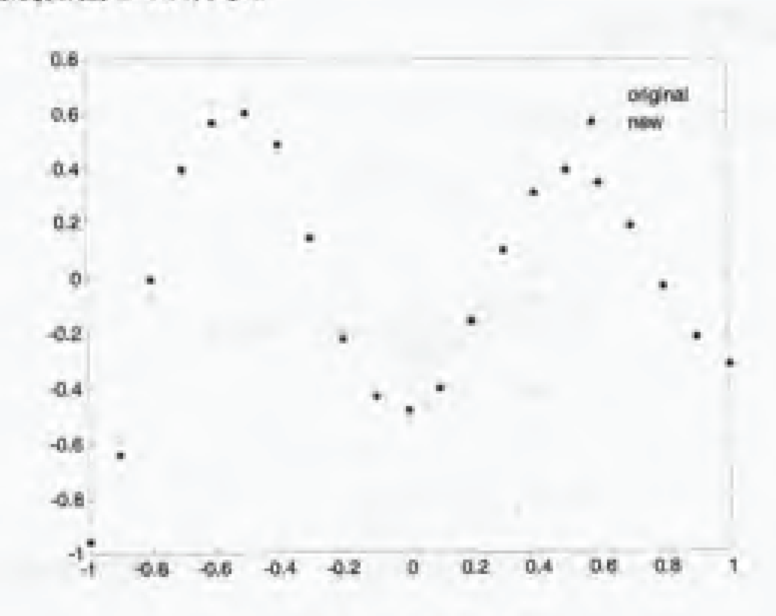


图 3-15 函数逼近结果

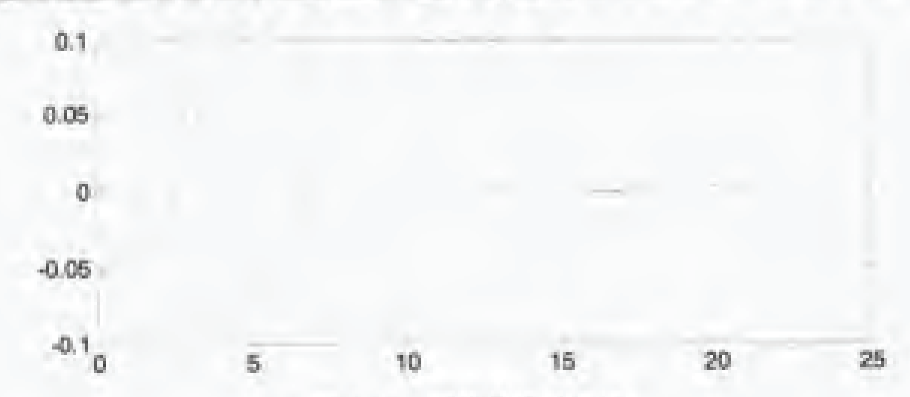


图 3-16　网络的误差曲线

至此，我们确定了本例的最终 BP 网络结构，如表 3-3 所示。

表 3-3　网络最终结构

网络结构	隐含层神经元	训练函数	网络误差
单隐层的 BP 网络	8 个	trainlm	0.1444

3.3　线性神经网络及 MATLAB 实现

线性神经网络和感知器网络在结构上比较相似，区别在于线性神经网络的传递函数为线性的，而感知器的传递函数为二值型的。同感知器网络一样，线性神经网络也只能对线性可分模式进行分类。但是，相对于感知器网络来说，线性神经网络的学习算法的收敛速度和精度都有较大程度的提高。

20 世纪 50 年代末，由 Widrow 和 Hoff 提出的自适应线性元件是线性神经网络最早的，也是最典型的代表。

3.3.1　线性神经网络的结构

线性神经网络的神经元结构如图 3-17 所示。

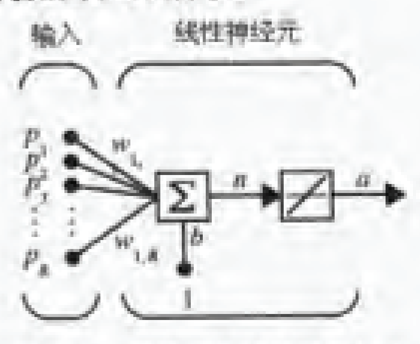


图 3-17　线性网络的神经元结构

由图 3-17 也可以看出，线性神经网络的神经元结构与感知器的神经元结构的差异只在于传递函数的不同。线性神经网络的传递函数为 $f(x)=x$。

一个由 S 个神经元构成的线性神经网络结构如图 3-18 所示。由图可见，该网络有 R 个输入向量。这种网络称为 Madaline 网络。

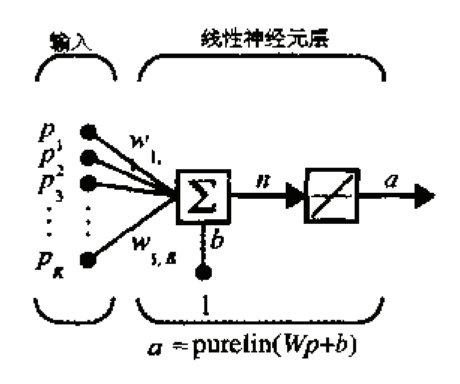


图 3-18　线性神经网络结构

虽然这只是一个单层线性网络，但它的功能和多层线性网络是一样的。任何一个多层线性网络，都对应着一个与之等价的单层线性网络。

3.3.2　线性神经网络的学习

本节以 Madaline 网络为例，介绍线性网络的学习过程。Madaline 网络的学习过程是按照“误差平方和最小”的原则，即 LMS（Least Mean Square）算法，反复对各连接权值进行修正的过程。这里的误差指的是实际输出和目标向量之间的差值。这种学习过程所使用的规则称为 Widrow-Hoff 学习规则。

令 $P_k=\left(p_1^k,p_2^k,\cdots,p_R^k\right)$ 表示网络的输入向量，$T_k=\left(y_1^k,y_2^k,\cdots,y_s^k\right)$ 表示网络的目标向量。其中，$k=1,2,\cdots,m$，m 为学习模式（一个输入向量与对应的目标向量称为一个学习模式）的数目，s 为输出层单元的数目。$W_{ij}=\left(w_{i1},w_{i2},\cdots,w_{is}\right)$ 表示连接权值向量，其中，$i=1,2,\cdots,R$。学习过程如下：

（1）初始化。为各连接权值 w_{ij} 赋予区间[-1，1]内的随机值；

（2）任选一组学习模式提供给网络；

（3）计算网络输出值：

$$z_j^k=\sum_{i=0}^{R}w_{ij}p_i \qquad j=1,2,\cdots,s$$

（4）计算网络各输出单元的实际输出与目标向量之间的误差：

$$d_j^k=y_j^k-z_i^k \qquad j=1,2,\cdots,s$$

（5）进行连接权值的修正：

$$w_{ij}(N+1)=w_{ij}(N)+\alpha a_i^k d_j^k \quad i=1,2,\cdots,R$$

（6）取下一个学习模式提供给网络，重复步骤（3）～（5），直到误差d_j^k变得足够小为止。



Madaline 的 LMS 学习规则可保证最终的误差函数最小，但这需要进行无限次的学习，因此实际应用中只能得到近似值。

3.3.3 线性网络的 MATLAB 仿真

例 3.2 这里给出一个输入向量 P 和目标向量 T，P=[1.0 –1.3]，T=[0.5 1.0]，要求通过设计一个线性网络来寻找两者之间的线性关系。

首先利用函数 newlin 创建一个线性网络。

```
P=[1 -1.3];
T=[0.5 1];
net = newlin(minmax(P),1,0,0.01);
```

上式表示网络只有输出，学习速率为 0.01。接下来需要对网络进行训练，设定网络的训练次数为 500 次，目标误差为 0，训练代码为：

```
net=init(net);
net.trainParam.epochs = 500;
net = train(net,P,T);
```

训练结果为：

```
TRAINB, Epoch 0/500, MSE 0.625/1e-005.
TRAINB, Epoch 25/500, MSE 0.199208/1e-005.
TRAINB, Epoch 50/500, MSE 0.0667243/1e-005.
TRAINB, Epoch 75/500, MSE 0.0232928/1e-005.
TRAINB, Epoch 100/500, MSE 0.00839361/1e-005.
TRAINB, Epoch 125/500, MSE 0.00309461/1e-005.
TRAINB, Epoch 150/500, MSE 0.00115901/1e-005.
TRAINB, Epoch 175/500, MSE 0.000438647/1e-005.
TRAINB, Epoch 200/500, MSE 0.000167147/1e-005.
TRAINB, Epoch 225/500, MSE 6.3971e-005/1e-005.
TRAINB, Epoch 250/500, MSE 2.45514e-005/1e-005.
TRAINB, Epoch 274/500, MSE 9.80678e-006/1e-005.
TRAINB, Performance goal met.
```

可见，网络利用函数 trainb 进行训练，在经过 274 次训练后，网络的误差达到要求，训练结果如图 3-19 所示。

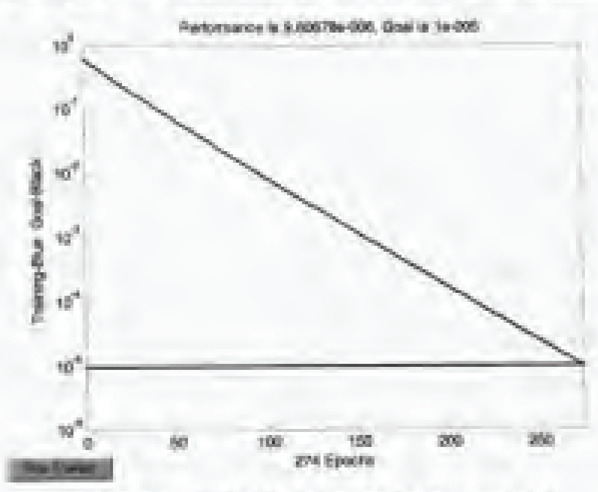


图 3-19　训练结果（训练函数：trainb）

在命令行窗口中输入：

```
y=sim(net,P)
```

按回车键后，结果为：

```
y =
    0.4960    0.9982
```

和目标向量对比，误差比较小。

实际上，P 和 T 之间的线性关系可以通过方程 $T=W\times P+B$ 来描述，所以 W 和 B 可以通过解下面的方程得出：

$$\begin{cases} W+B=0.5 \\ -1.3W+B=1.0 \end{cases}$$

则 $W=-0.2173$，$B=0.7173$，W 和 B 分别对应着网络的权值和阈值，在命令行窗口中输入：

```
w=net.IW{1,1}
b=net.b{1}
```

结果为：

```
w =
    -0.2183
b =
    0.7143
```

此结果与通过解方程得到的 W 和 B 是很接近的，所以，上面的网络可以很好地实现一个简单的线性关系。

例 3.3　例 3.2 是对一个比较小的输入向量和目标向量的模式联想的设计问题，接下来提供一个较大的多神经元网络的模式联想的设计问题，输入向量 P 和目标向量 T 分别为：

```
P=[1 1.5 1.2 -0.3;-1 2 3 -0.5;2 1 -1.6 0.9];
T=[0.5 3 -2.2 1.4;1.1 -1.2 1.7 -0.4;3 0.2 -1.8 -0.4;-1 0.1 -1.0 0.6];
```

由 P 和 T 可得，网络为 3×4 的结构，即网络的输入层有 3 个神经元，输出层有 4 个神经元，两层之间的神经元为全连接。

和例 3.2 一样，本例的问题也可以采用线性方程组求解，即针对每个输出结点写出输

入和输出之间的关系式。对于网络，每个输出神经元都有 4 个变量，包括 3 个权值和 1 个阈值，同时有 4 个限制方程，每组输入对应 4 个输出值。这样一来，4 个输出结点就可以得到 16 组方程。因此，只要输入向量是线性不相关的，则同时满足这 16 个方程的权值存在且惟一，对应到网络，就是网络可以通过训练找到零误差的精确权值。

通过以下代码创建并训练一个线性网络：

```
net = newlin(minmax(P),4,0,0.01);
net=init(net);
net.trainParam.epochs = 2000;
net.trainParam.goal=0.0001;
net = train(net,P,T);
```

通过上述代码，设置网络的训练步数为 2000 次，训练的目标误差为 0.0001。代码的运行结果为（部分结果）：

```
......
TRAINB, Epoch 1275/2000, MSE 0.000109871/0.0001.
TRAINB, Epoch 1289/2000, MSE 9.94395e-005/0.0001.
TRAINB, Performance goal met.
```

可见，网络经过 1289 次训练后，误差达到目标误差的要求。训练的误差曲线如图 3-20 所示。

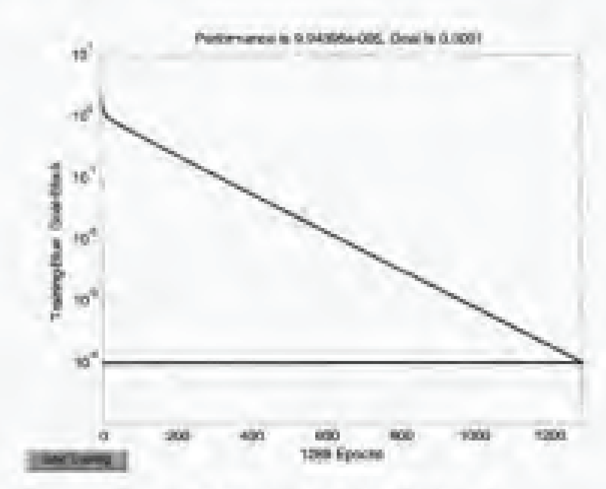


图 3-20　网络训练误差曲线

在命令行窗口中输入如下代码对网络进行仿真：

```
y=sim(net,P)
```

输出结果为：

```
y =
     0.5174     2.9848    -2.1887     1.3892
     1.0862    -1.1879     1.6910    -0.3915
     2.9925     0.2065    -1.8049    -0.3953
    -0.9917     0.0927    -0.9946     0.5949
```

将输出结果 y 与网络的目标向量 T 相比较，计算网络的误差。

```
x=y - T;
for i=1:4
    error(i)=norm(x(:,i))
```

```
end
```

可得到网络对于目标向量列的误差为：

```
error =
    0.0249    0.0218    0.0162    0.0154
```

可见网络的误差是非常小的。接下来，看一下网络的权值和阈值。

```
w=net.IW{1,1}
b=net.b{1}
```

结果为：

```
w =
   - 2.4593    2.2829    3.1516
     2.1796  - 1.8057  - 2.0533
     2.0800  - 1.2588    0.0494
   - 1.6810    0.9701    0.9853
b =
   - 1.0436
     1.2075
   - 0.4453
   - 0.3112
```

如果每个神经元的自由度（即权值和阈值数目）等于或大于限制数（即输入/输出向量对），那么线性网络就可以"零误差"地解决问题。不过，如果输入向量线性相关或者没有阈值时，该结论就不再成立。接下来就研究这两种特殊情况。

例 3.4 设计并训练一个线性网络，实现从输入向量 P 到输出向量 T 的转换。

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
```

从两个向量的结构来看，所设计的线性网络是具有阈值的，并且输入层的神经元为 2 个，输出层的神经元为 1 个。如果仔细观察，就会发现输入向量 P 是线性相关的。在这种情况下，线性网络能够解决这个问题。代码为：

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
net = newlin(minmax(P),1,0,0.01);
y=sim(net,P);
net=init(net);
net.trainParam.epochs = 2000;
net.trainParam.goal=0.0001;
net = train(net,P,T);
```

网络的训练结果为：

```
TRAINB, Epoch 1000/2000, MSE 0.214287/0.0001.
TRAINB, Epoch 1025/2000, MSE 0.214286/0.0001.
……
TRAINB, Epoch 1975/2000, MSE 0.214286/0.0001.
TRAINB, Epoch 2000/2000, MSE 0.214286/0.0001.
```

可见，网络训练到第 1000 步至 1500 步时，网络误差发生了轻微的改变，从 1500 步后，网络误差就不再改变，如图 3-21 所示，此时的网络训练误差为 0.2143 左右。

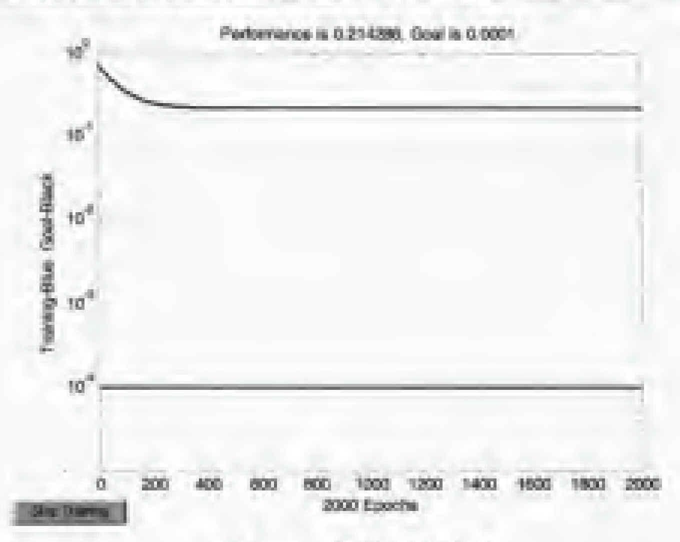


图 3-21 训练误差曲线

对网络进行仿真，得到此时的网络输出 Y。

```
Y=sim(net,P)
```

可得 Y=0.9286　　0.3571　　－0.7857，与目标向量 T 相比较，发现误差还是比较大的。这说明在这种情况下，网络是无法给出完美的解决方案的。

例 3.5 给定如下的输入向量 P 和输出向量 T，再次利用线性网络求解两者之间的关系。本例的目的在于了解线性网络的线性逼近求解的能力。

```
P=[1 1.5 3.0 -1.2 ];
T=[0.5 1.1 3.0 -1.0 ];
```

此时，问题相当于用 4 个方程解 w 和 b 两个未知数，由于方程数目大于未知数的数目，所以直接解方程是无解的，但仍可利用线性网络解决这个问题。从两个向量的结构可以看出，此时，网络的结构为 1 个输入神经元和 1 个输出神经元，1 个权值 w 和 1 个阈值 b。

首先将 P 和 T 绘制在一起，代码为：

```
plot(P,T,'+');
axis([-1.5 3.5 -1.5 3.5]);
xlabel('training vector P');
ylabel('target vector T');
```

绘制结果如图 3-22 所示。接下来创建一个线性网络，并进行训练。代码为：

```
net = newlin(minmax(P),1,0,0.01);
net=init(net);
net.trainParam.epochs = 500;
net.trainParam.goal=0.0001;
net = train(net,P,T);
```

训练结果如图 3-23 所示。

```
TRAINB, Epoch 300/500, MSE 0.0723588/0.0001.
TRAINB, Epoch 325/500, MSE 0.0723587/0.0001.
……
TRAINB, Epoch 475/500, MSE 0.0723587/0.0001.
TRAINB, Epoch 500/500, MSE 0.0723587/0.0001.
TRAINB, Maximum epoch reached.
```

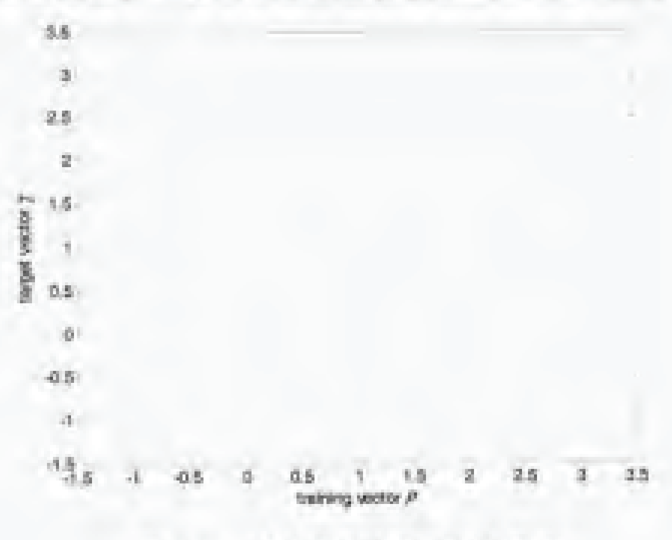


图 3-22 输入向量 P 和目标向量 T

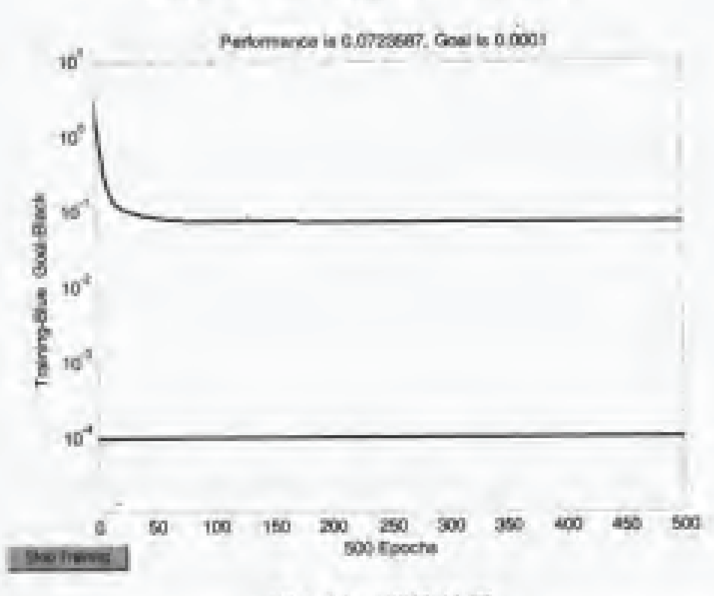


图 3-23 训练结果

由此可见，网络自第 325 次训练后，误差就不再改变，此时的 MSE= 0.0723587，误差还是比较大的。在命令行窗口中输入：

```
y=sim(net,P)
```

得到 y=0.8299　　1.2975　　2.7003　　- 1.2276。

最后检验网络是否很好地逼近了 P 和 T 之间的线性关系。代码为：

```
plot(P,y);
hold on
plot(P,T,'r+');
```

运行结果如图 3-24 所示。从图中可以看出，网络的逼近误差是比较大的。因此，可以得出如下结论：线性网络只可以学习输入/输出向量之间的线性关系。所以，对于某些特殊的问题，网络无法得到满意的结果。但是，即使不存在一个完美的结果，只要学习速率足够小，对于给定的结构，线性网络总可得到一个接近目标的结果。

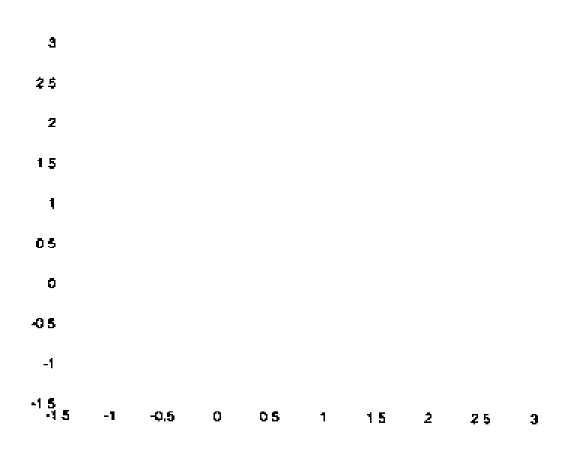


图 3-24　网络逼近曲线

3.4　径向基函数网络及 MATLAB 实现

径向基函数 RBF 神经网络（简称径向基网络）是由 J.Moody 和 C.Darken 于 20 世纪 80 年代末提出的一种神经网络结构，它是具有单隐层的三层前馈网络。目前已经证明，径向基网络也能够以任意精度逼近任意连续函数。

3.4.1　径向基网络结构

如果要实现同一个功能，径向基网络的神经元个数可能要比前向 BP 网络的神经元个数要多，但是，径向基网络所需要的训练时间却比前向 BP 网络的要少。

径向基网络的神经元模型结构如图 3-25 所示。由图可见，径向基网络传递函数 radbas 是以权值向量和阈值向量之间的距离||dist||作为自变量的，其中，||dist||是通过输入向量和加权矩阵的行向量的乘积得到的。

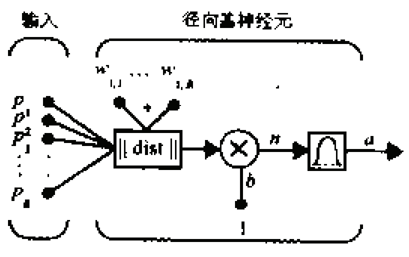


图 3-25　径向基神经元模型结构

径向基网络传递函数的原型函数为：

$$\mathrm{radbas}\left(n\right)=\mathrm{e}^{-n^2}$$

当输入自变量为 0 时，传递函数取得最大值为 1。随着权值和输入向量之间距离的减少，网络输出是递增的。所以，径向基神经元可以作为一个探测器，当输入向量和加权向量一致时，神经元输出 1。图 3-25 中的 b 为阈值，用于调整神经元的灵敏度。

利用径向基神经元和线性神经元可以建立广义回归（Generalized Regression）神经网络，这种形式的网络经常用于函数逼近。广义回归神经网络结构如图 3-26 所示。该网络包括两层，中间层（隐含层）为径向基层，输出层为线性层。

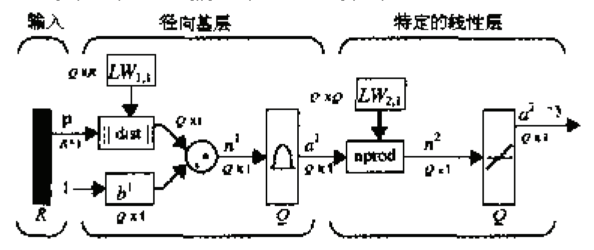


图 3-26 广义回归神经网络结构

图 3-26 所示的广义回归神经网络有 Q 组输入向量，每组向量的元素个数为 R 个，中间层有 S^1 个径向基神经元，输出层有 S^2 个线性神经元。

径向基神经元还可以和竞争神经元一起共同组建概率神经网络（Probabilistic Neural Network, PNN）。概率神经网络经常用于解决分类问题。网络结构如图 3-27 所示。

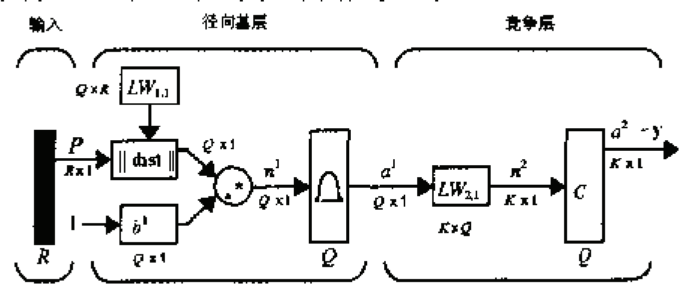


图 3-27 概率神经网络结构

概率神经网络按此方式进行分类：为网络提供一种输入模式向量后，首先，径向基层计算该输入向量同样本输入向量之间的距离||dist||，该层的输出为一个距离向量。竞争层接受距离向量为输入向量，计算每个模式出现的概率，通过竞争传递函数为概率最大的元素对应输出 1，这就是一类模式；否则输出 0，作为其他模式。

3.4.2 径向基函数的学习过程

径向基函数网络是由输入层、隐含层和输出层构成的三层前向网络（以单个输出神经元为例），隐含层采用径向基函数作为激励函数，该径向基函数一般为高斯函数，如图 3-28 所示。隐层每个神经元与输入层相连的权值向量 $W1_i$ 和输入矢量 X^q（表示第 q 个输入向量），之间的距离乘上阈值 $b1_i$ 作为本身的输入，如图 3-29 所示。

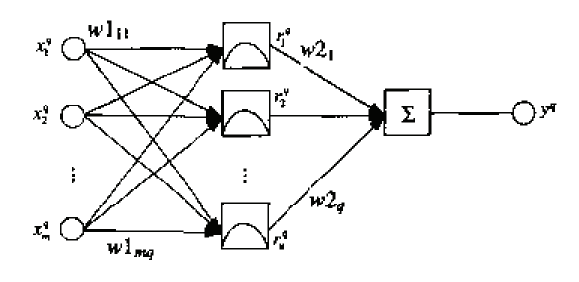


图 3-28 RBF 网络结构

由此可得隐含层的第 i 个神经元的输入为：

$$k_i^q = \sqrt{\sum_j \left(w1_{ji} - x_j^q\right)^2} \times b1_i$$

输出为：

$$r_i^q = \exp\left(-\left(k_i^q\right)^2\right) = \exp\left(\sqrt{\sum_j \left(w1_{ji} - x_j^q\right)^2} \times b1_i\right) = \exp\left(-\left(\| w1_i - X^q \| \times b1_i\right)^2\right)$$

图 3-29　RBF 网络隐含层神经元的输入与输出

径向基函数的阈值 $b1$ 可以调节函数的灵敏度，但实际工作中更常用另一参数 C（称为扩展常数）。$b1$ 和 C 的关系有多种确定方法，在 MATLAB 神经网络工具箱中，$b1$ 和 C 的关系为 $b1_i$=0. 8326/ C_i，此时隐含层神经元的输出变为：

$$g_i^q = \exp\left(\frac{\sqrt{\sum_j \left(w1_{ji} - x_j^q\right)^2} \times 0.8326}{C_i}\right) = \exp\left(-0.8326^2 \times \left(\frac{\| w1_i - X^q \|}{C_i}\right)^2\right)$$

由此可见，C 值的大小实际上反映了输出对输入的响应宽度。C 值越大，隐含层神经元对输入矢量的响应范围将越大，且神经元间的平滑度也较好。

输出层的输入为各隐含层神经元输出的加权求和。由于激励函数为纯线性函数，因此输出为：

$$y^q = \sum_{i=1}^{n} r_i \times w2_i$$

RBF 网络的训练过程分为两步：第一步为无教师式学习，确定训练输入层与隐含层间的权值 $w1$；第二步为有教师式学习，确定训练隐含层与输出层间的权值 $w2$。在训练以前，需要提供输入矢量 X、对应的目标矢量 T 与径向基函数的扩展常数 C。训练的目的是求取两层的最终权值 $w1$、$w2$ 和阈值 $b1$、$b2$（当隐含层单元数等于输入矢量数时，取 $b2 = 0$）。

在 RBF 网络训练中，隐含层神经元数量的确定是一个关键问题，传统的做法是使其与输入向量的元素相等。显然，在输入矢量很多时，过多的隐含层单元数是难以让人接受的。为此，我们提出了改进方法，基本原理是从 0 个神经元开始训练，通过检查输出误差使网络自动增加神经元。每次循环使用，使网络产生的最大误差所对应的输入向量作为权值向

量 $w1_i$，产生一个新的隐含层神经元，然后检查新网络的误差，重复此过程直到达到误差要求或最大隐含层神经元数为止。由此可见，径向基函数网络具有结构自适应确定、输出与初始权值无关等特点。

3.4.3　RBF 网络应用实例

这里利用 RBF 网络结构自适应确定、输出与初始权值无关等优良特性，将该网络应用于某地的地下水动态模拟与预测，演示训练样本集与检测样本集的构建、原始数据的预处理、神经网络的构建训练和检测及结果评价的整个过程。

1. 前期准备

地下水位主要受河道流量、气温、饱和差、降水量和蒸发量等重要因子影响，由此从资料中归纳出 24 组数据，如表 3-4 所示。选定其中的 1～19 组作为训练样本，20～24 组作为测试样本。

表 3-4　地下水位及其影响因子监测数据表

序　号	河道流量	气　温	饱和差	降水量	蒸发量	水　位
1	0.0177	0	0.0200	0.0054	0.0580	0.6725
2	0.0230	0	0.1000	0.0054	0	0.6943
3	0.0619	0.2424	0.1500	0.0323	0.2319	0.6376
4	0.2212	0.6061	0.4000	0.1613	0.5217	0.4891
5	0.0796	0.8182	0.8000	0.0968	0.7971	0.1616
6	0.1504	0.9697	0.9000	0.6075	0.8406	0.0699
7	0.1681	1.0000	0.7000	0.1559	0.6957	0.1092
8	0.1504	0.9394	0.5000	0.3978	0.5507	0.1048
9	0.1150	0.7576	0.4000	0.1129	0.2174	0.2533
10	0.1593	0.5606	0.4000	0.0806	0.3913	0.4017
11	0.0885	0.3030	0.5200	0.0753	0.2319	0.6201
12	0.2035	0.3182	0.3500	0.0591	0	0.6638
13	0	0.3333	0.1000	0.0054	0.0290	0.5764
14	1.0000	0.9697	0.4500	1.0000	0.6812	0.0437
15	0.6106	0.8788	0.4000	0.6129	0.5507	0
16	0.6814	0.6970	0.4000	0.3226	0.4058	0.0568
17	0.3982	0.4848	0.2000	0.1882	0.2609	0.2009
18	0.1858	0.3333	0.1000	0.0215	0.1304	0.4105
19	0.0708	0.0909	0	0.0323	0.0290	0.5153
20	0.0442	0.0909	0.1500	0.0108	0.0725	0.6070
21	0.1150	0.3030	0.2000	0.0215	0.4783	1.0000
22	0.1681	0.6061	0.6000	0	0.3478	0.4148
23	0.0708	0.8485	0.9000	0.1022	0.8261	0.1921
24	0.1327	0.9545	1.0000	0.4355	1.0000	0.0873



获得有关地下水位的数据后，在用于训练样本和测试样本之前，需要进行归一化处理。表 3-4 中的数据为已经归一化处理的数据。

2. 网络的创建、训练和测试

RBF 网络的输入层神经元个数取决于地下水位影响因子的个数，由表 3-4 可知，其个数为 5。由于输出就是地下水位的值，所以输出层神经元个数为 1。利用函数 newrbe 创建一个精确的神经网络，该函数在创建 RBF 网络时，自动选择隐含层的数目，使得误差为 0。代码为：

```
SPREAD=1.5;
net = newrbe(P,T,SPREAD);
```

其中，P 为输入向量，T 为目标向量，它们可从表 3-4 中得到。SPREAD 为径向基函数的分布密度，SPREAD 越大，函数越平滑，这里先取 1.5。由于网络的建立过程就是训练过程，因此，此时得到的网络 net 已经是训练好了的。

接下来对网络进行仿真，验证其预测性能。代码为：

```
y=sim(net,P_test)
```

其中，P_test 为网络的测试样本，从表 3-4 中得出。运行结果为：

```
y =
     0.6455     1.0844     0.3816     0.0064     0.1837
```

经过反归一化处理，可得地下水位 H=6.8581　7.8632　6.2538　5.3947　5.8007。

同实际值 $H0$=6.777.676.335.82 5.58 相比较，可得出预测误差如图 3-30 所示。由图可见，对于地下水位的预报来说，网络的预报误差并不大。

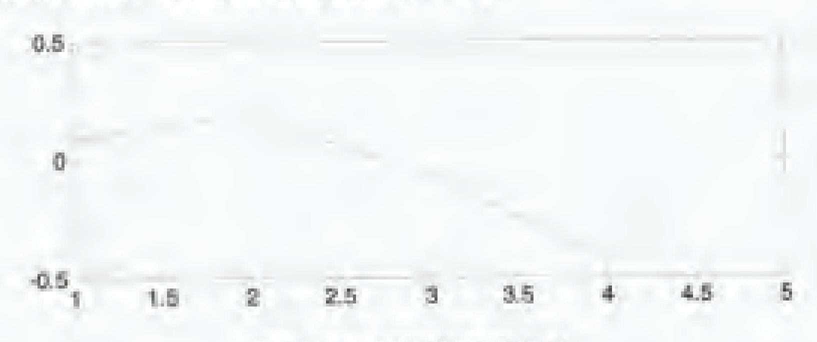


图 3-30　网络的预测误差

此外，SPREAD 值的大小影响网络的预测精度。接下来，分别在 SPREAD=2,3,4,5 的情况下计算网络的预报精度。代码为：

```
y=rands(4,5);
for i=1:4
    net = newrbe(P,T,i+1);
    y(i,:)=sim(net,P_test);
end
plot(1:5,y(1,:)-T_test,'r');
hold on
plot(1:5,y(2,:)-T_test,'b-');
hold on
plot(1:5,y(3,:)-T_test,'g+');
```

```
hold on
plot(1:5,y(4,:)-T_test,'.');
hold on
y_bp=[0.1201 -0.2559 -0.1828 -0.1237 0.0001];
plot(1:5,y_bp,'*');
hold off
```

结果如图3-31所示。可以看出，当SPREAD=2或3时，网络的预报误差最小。因此，本例中SPREAD=2或3都可得到理想的结果。图中“*”号表示的为采用BP网络进行地下水位预报的误差。

为了验证RBF网络相对于BP网络的优势，在本例中利用BP网络对地下水位进行重新预报。选择的BP网络为5×11×1的结构，训练函数为trainlm。经过500次训练后，对网络进行仿真，并计算网络的预测误差。误差变量用y_bp表示。综合对比后发现，对于预报精度来说，BP网络明显不如RBF网络，而且BP网络的训练时间明显大于RBF网络，其训练速度比较慢。

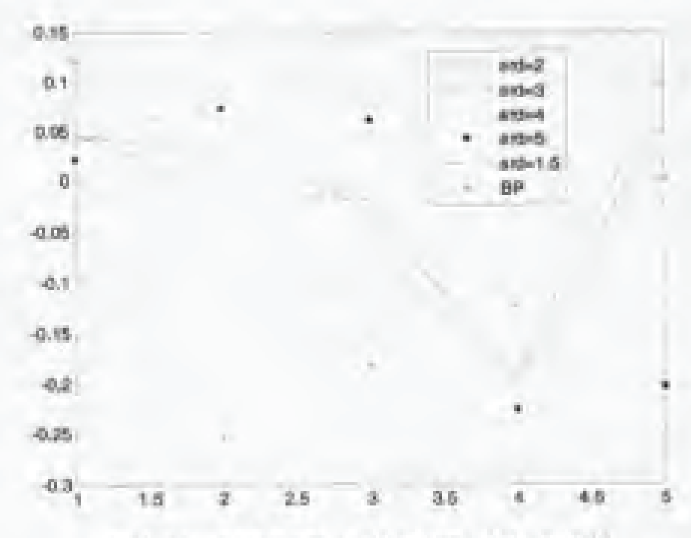


图3-31　SPREAD取不同值时的预报误差

3.4.4　基于RBF网络的非线性滤波

1. 非线性滤波

早期的数字信号处理和数字图像处理主要以线性滤波器为主要处理手段。线性滤波器由于数学表达式比较简单并且具有其他一些比较理想的特性，所以，实现起来相对比较容易。然而，当信号中存在由系统非线性引起的噪声或非高斯叠加型噪声时，线性滤波器便不能很好地工作。目前，最优非线性滤波存在“实时”问题，即：（1）滤波器权系数的实时计算；（2）非线性滤波器的实时实现。描述系统的非线性差分方程为：

状态方程：$x(n+1)=f\left(x(n)\right)+v(n)$；

观测方程：$y(n+1)=h\left(x(n)\right)+w(n)$。

其中，f 和 h 都是非线性函数，$w(n)$和 $v(n)$为零均值的白噪声序列。

所谓最优滤波，就是解决从观测值 $y(n)$估计出状态 $\hat{x}(n)$，且使得 $\hat{x}(n)$ 可以最好地接近 $x(n)$。RBF 网络具有惟一的最佳逼近特性，因此尝试将其应用于最优滤波，即利用已知的采样数据对非线性函数做最佳逼近。由 RBF 网络的输入/输出表达式可得 h 的估计值：

$$\hat{h}=\sum_{i=1}^{N} w_i R_i(\cdot)=W^{\mathrm{T}} r(\cdot)$$

其中，$W=\left[w_i\right]_{i=1}^{N}$，$r=\left[R_i(\cdot)\right]_{i=1}^{N}$，$N$ 为训练次数。接下来，设计一个 RBF 网络，使得它可以在规定的精度内逼近 h。

2. 网络设计

输入样本为 P，目标向量为 T，如下所示：

```
P=[-1:0.1:1];
T=[-0.9602 0.5770 0.0729 0.3771 0.6405 0.6600 0.4609 0.1336 -0.2013 -0.4344 -0.5000 -0.3930
0.1647 0.0988 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
for i=1:5
    net=newrbe(P,T,i);
    y(i,:)=sim(net,P);
end
```

在上面的代码中，我们利用 RBF 网络精确创建函数 newrbe，创建了一个准确的 RBF 网络，它已经可以逼近目标向量了。由于径向基函数的分布密度 SPREAD 可以影响网络的精度，因此，这里将其设定为 1,2,3,4 和 5，共 5 个整数，观察它们对网络预测性能的影响。

网络的逼近误差如图 3-32 所示，由图可见，分布密度为 1 和 2 时，网络的逼近误差比较小，考虑到收敛速度和计算方面的原因，这里的分布密度选 1。

此时网络的输出结果为：

```
y(1,:)=
  -0.9587    0.5671    0.1014    0.3379    0.6597    0.6715    0.4481    0.1320
 -0.1972   -0.4463   -0.4964   -0.3111   -0.0089    0.2262    0.3167
  0.3365    0.3448    0.2322   -0.0725   -0.2044   -0.3221
```

本实例的完整 MATLAB 代码为：

```
P=[-1:0.1:1];
T=[-0.9602 0.5770 0.0729 0.3771 0.6405 0.6600 0.4609 0.1336 -0.2013 -0.4344 -0.5000 -0.3930
0.1647 0.0988 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
%创建 5 个 RBF 网络，分布密度分别为 1,2,3,4,5
for i=1:5
    net=newrbe(P,T,i);
    y(i,:)=sim(net,P);
end
%绘制误差曲线
plot(1:21,y(1,:)-T);
hold on;
plot(1:21,y(2,:)-T,'+');
```

```
hold on;
plot(1:21,y(3,:)-T,'.');
hold on;
plot(1:21,y(4,:)-T,'r--');
hold on;
plot(1:21,y(5,:)-T,'g-.');
hold off;
```

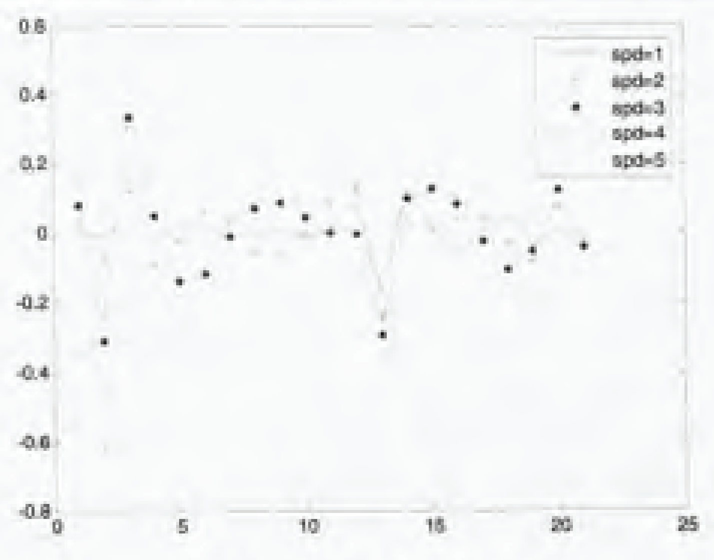


图 3-32　网络的逼近误差曲线

3.4.5　基于 GRNN 的函数逼近

广义回归神经网络 GRNN 是径向基网络的一种变化形式，由于训练速度快，非线性映射能力很强，因此它经常用于函数逼近。

接下来设计一个简单的 GRNN 网络，用于对非线性函数的逼近。

1. 问题描述

这里要逼近的函数为指数函数 $y=\exp(-x)$（$-1\leqslant x<1$），绘制该函数的曲线，如图 3-33 所示。

```
p=-1:0.05:1;
t=exp(-p);
plot(p,t);
grid;
title('exponential function');
xlabel('x');
xlabel('y');
```

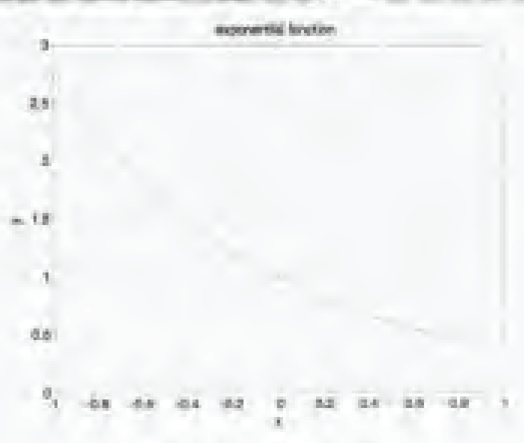


图 3-33 指数函数

2. 网络创建与训练

径向基函数的分布密度 SPREAD 可以对 GRNN 的性能产生重要影响。理论上讲，SPREAD 越小，对函数的逼近就越精确，但是逼近的过程就越不平滑；SPREAD 越大，逼近过程就比较平滑，但是逼近误差会比较大。

由于 SPREAD 的大小对网络的最终逼近精度有着比较大的影响，因此在网络设计过程中需要调整 SPREAD 的值，直到达到比较理想的精度。

接下来将 SPREAD 分别设置为 0.1,0.2,0.3,0.4 和 0.5，检验不同的 SPREAD 值对网络逼近性能的影响。

```
for i=1:5
    net=newgrnn(p,t,i/10);
    y(i,:)=sim(net,p);
end
```

此时网络的输出与对函数的逼近效果如图 3-34 所示。由图可见，当 SPREAD=0.1 时，网络对函数的逼近效果最好；当 SPREAD=0.5 时，网络的逼近效果最差。这也验证了上面的结论。

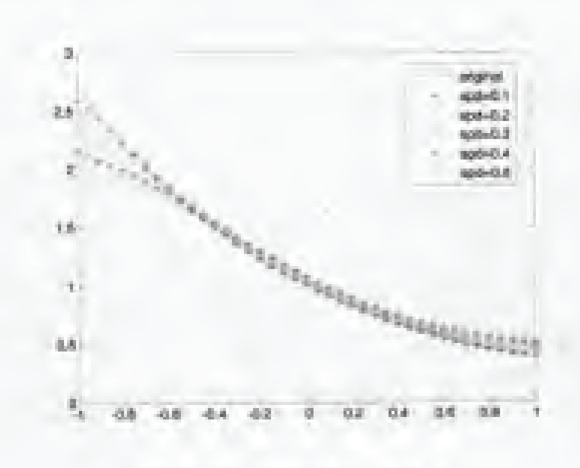


图 3-34 网络的逼近结果

因此，可将 SPREAD 的最佳值设定为 0.1，此时网络对函数的逼近误差如图 3-35 所示。由图可见，网络的逼近误差一直在 0 的附近波动，这说明网络对函数的逼近效果是非常好的。

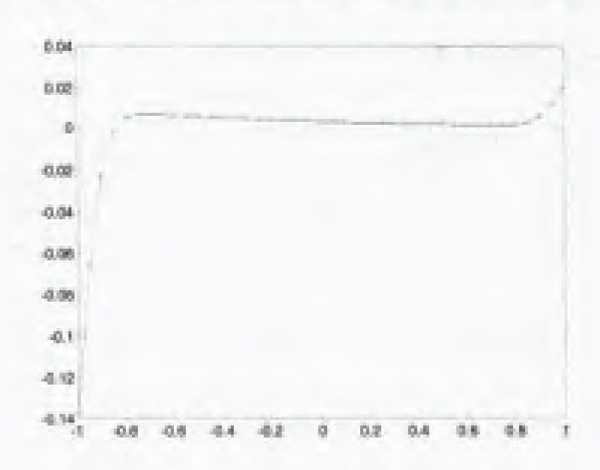


图 3-35　网络的逼近误差

本例的完整 MATLAB 代码为：

```
%绘制指数函数曲线
p=-1:0.05:1;
t=exp(-p);
plot(p,t);
grid;
title('exponential function');
xlabel('x');
ylabel('y');
figure;
%建立并训练网络
for i=1:5
    net=newgrnn(p,t,i/10);
    y(i,:)=sim(net,p);
end
%绘制网络的逼近效果
plot(p,t);
hold on;
plot(p,y(1,:),'*');
hold on;
plot(p,y(2,:),'d');
hold on;
plot(p,y(3,:),'ro');
hold on;
plot(p,y(4,:),'kp');
hold on;
plot(p,y(5,:),'g+');
hold off;
figure;
%绘制网络的最佳逼近误差
plot(p,y(1,:)-t,'+-');
```

3.4.6 基于概率神经网络的分类

概率神经网络 PNN 也是径向基网络的一种变化形式，它具有结构简单、训练快捷等特点，应用非常广泛，特别适合于模式分类问题的解决。在模式分类中，它的优势在于可以利用线性学习算法来完成以往非线性算法所做的工作，同时又可以保持非线性算法高精度的特性。

本节以一个非常简单的例子说明 PNN 在分类问题中的应用。

1. 问题描述

假定输入为一个一维向量，一共有 7 个元素：

```
P = [1 2 3 4 5 6 7];
```

该向量中这些元素所属的类别为：

```
Tc = [1 2 3 2 2 3 1];
```

利用函数 plotvec 绘制出这些元素的类别：

```
plotvec(P,Tc);
```

结果如图 3-36 所示。

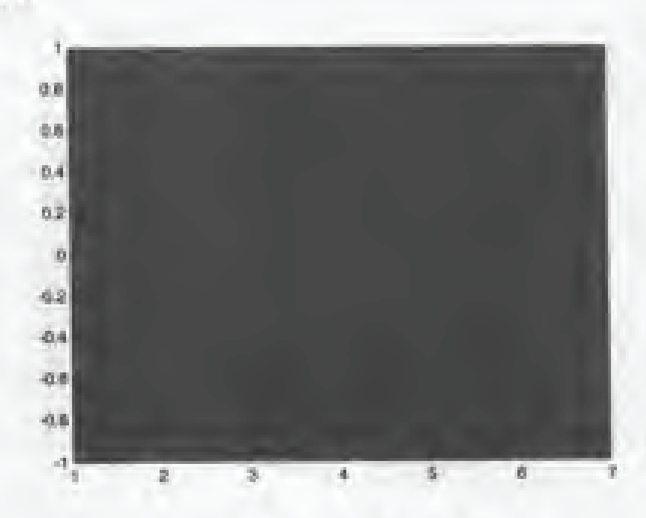


图 3-36 输入向量结果

plotvec 可以根据第 2 个参数的情况，利用不同颜色绘制出第 1 个参数中相应的元素，读者可以在 MATLAB 命令行窗口中输入以上代码，然后观察结果。

2. 网络创建与训练

同 RBF 网络一样，径向基函数的分布密度 SPREAD 能够对网络的分类性能产生比较严重的影响。当 SPREAD 接近于 0 时，对应的 PNN 就成为一个最邻域分类器；当 SPREAD 增大后，对应的 PNN 就要考虑附近的设计向量。

这里将 SPREAD 设定为 0.1，创建一个 PNN。

```
T = ind2vec(Tc)
net = newpnn(P,T);
Y = sim(net,P)
Yc = vec2ind(Y)
```

SPREAD=0.1 为函数 newpnn 的默认值，因此，这里无需特别设定。函数 ind2vec 将类别向量转换为 PNN 可以使用的目标向量 T；函数 vec2ind 将分类结果转换为容易识别的类别向量。

网络的输出结果为：

```
Y =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (2,4)        1
   (2,5)        1
   (3,6)        1
   (1,7)        1
Yc =
     1     2     3     2     2     3     1
```

可见网络对输入向量进行了正确分类。

本例完整的 MATLAB 代码为：

```
P = [1 2 3 4 5 6 7];
Tc = [1 2 3 2 2 3 1];
plotvec(P,Tc);
T = ind2vec(Tc)
%创建一个 PNN，SPREAD=0.1
net = newpnn(P,T);
%仿真网络输出
Y = sim(net,P)
Yc = vec2ind(Y)
```

3.5 GMDH 网络及 MATLAB 实现

GMDH 的全称是 Group Method of Data Handling（数据处理的群方法），GMDH 网络也称为多项式网络，它是前馈神经网络中常用的一种用于预测的神经网络。它的特点是网络结构不固定，而是在训练过程中不断地改变。

3.5.1 GMDH 网络理论

GMDH 网络的结构在训练过程中是变化的，如图 3-37 所示的是训练后的一个比较典型的 GMDH 网络。

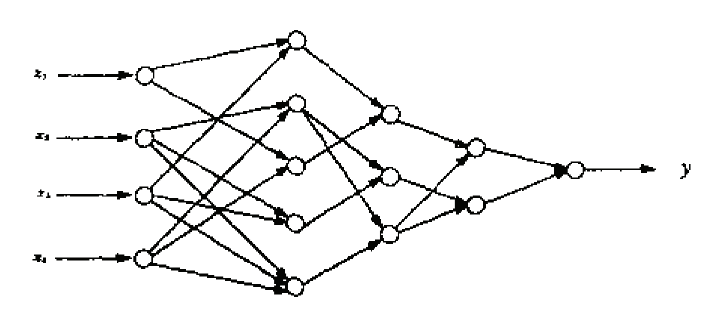


图 3-37　GMDH 网络结构

该网络有 4 个输入和 1 个输出。GMDH 网络的输入层将输入信号前向传递到中间层，中间层的每个神经元和前一层的 2 个神经元对应，因此，输出层的前一层（中间层）肯定只有 2 个神经元。

一般采用自适应线性元件作为 GMDH 网络中的神经元，如图 3-38 所示。该神经元的输入输出关系为：

$$Z_{k,l}=w_5Z_{k-1,i}^2+w_4Z_{k-1,i}Z_{k-1,j}+w_3Z_{k-1,j}^2+w_2Z_{k-1,i}+w_1Z_{k-1,j}+w_0$$

其中，$Z_{k,l}$ 表示第 k 层的第 l 个处理单元，且 $z_{0,l}=x_l$，$w_i\left(i=1,2,3,4,5\right)$ 为神经元的权值。由上式可见，GMDH 网络中的处理单元的输出是 2 个输入量的二次多项式，因此网络的每一层将使得多项式的次数增大 2 阶，其结果是网络的输出可以表示成输入的高阶（$2k$ 阶）多项式，其中 k 是网络的层数（不含输入层）。

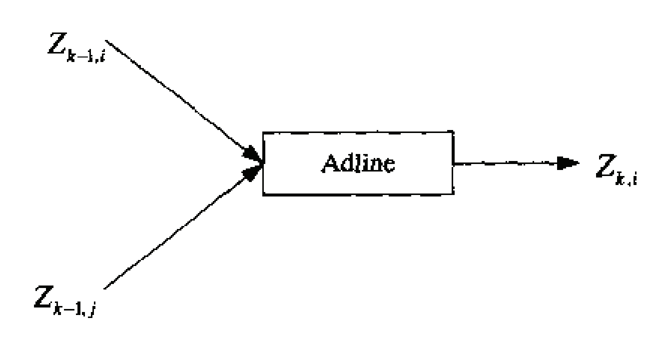


图 3-38　GMDH 网络中的神经元（输入层神经元除外）

3.5.2　GMDH 网络的训练

训练一个 GMDH 网络，包括从输入层开始构造网络，调整每一个神经元的权值和增加网络层数直到满足映射精度为止。第一层的神经元数取决于输入信号的数量，每一个输入信号需要一个神经元。一般地，假定网络仅有一个输入，所以输入层只有一个神经元。假设在时刻 k 神经元的权向量为：

$$w_k=\left[w_0,w_1,w_2,w_3,w_4,w_5\right]^{\mathrm{T}}$$

输入向量为：



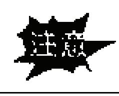
$$x_k = \left[1, x_1, x_1^2, x_1 x_2, x_2, x_2^2\right]$$

由 Widrow-Hoff 学习规则可知

$$w_{k+1}^{\mathrm{T}} = w_k^{\mathrm{T}} + \alpha \frac{x_k}{\left|x_k\right|^2}\left(y_{dk} - w_k x_k^{\mathrm{T}}\right)$$

其中，y_{dk} 为神经元在 k 时刻的目标输出向量，α 为学习速率，取值在 $[0.1,1)$ 之间。按照上式就可以调整神经元权值，降低神经元实际输出和目标输出之间的误差。

以上计算是在假定只有输入的前提下进行的。从权值调整公式可以看出，网络期望输出值 y_{dk} 出现在每个输入层神经元中，并希望通过训练使各神经元都能达到这一期望输出。对一个神经元来说，当训练数据集中每一个数据产生的均方差之和 S_E 达到最小时，对这个神经元的训练就结束，其权值予以固定。当输入层的神经元被全部训练一遍后，训练停止。这时，另一组数据（通常称为选择数据）被加到神经元上，并计算相应的 S_E。对那些 S_E 小于阈值的神经元，即长入下一层，而其余神经元则被舍弃，同时记录每一层神经元训练过程中产生的最小 S_E。若当前层在训练过程中产生的最小 S_E 小于前一层时（它表示网络精度得到提高），就产生一个新的神经元层，这一层中的神经元数取决于上一层中保留的神经元数，然后对新的神经元层进行训练和选择，而保持已训练的神经元层不变，这一过程一直进行到 S_E 不再减小为止，这时，取前一层神经元中误差最小的神经元的输出作为网络输出。当新神经元层只有 1 个神经元，且该层的 S_E 小于前一层时，这一神经元就作为输出神经元。输出神经元确定以后，要对网络进行整理，所有与输出神经元无直接或间接联系的神经元都被舍弃，仅留下那些与输出有关的神经元。图 3-37 即是一个训练结束后的 GMDH 网络。



在输入层中，输入为 x_1, x_4 的神经元已被舍去。而在第 2 层中，有 7 个神经元被舍弃。

3.5.3 基于 GMDH 网络的预测

一般来说，所有的神经网络均可用于预测。但是，对于一般的前馈网络来说，其结构（神经元层数及每层神经元个数）都是固定的，在训练过程中不会发生任何变化，因此一个网络的性能好坏与事前确定的该网络结构是否合适有很大关系。

与此不同的是，GMDH 网络的结构是在训练中动态确定的。在训练过程中，网络的神经元层数不断增加，每增加一层就增加一些新的神经元，而那些性能不好的神经元则被舍弃，因而每一层中的神经元数也是可变的。

下面是训练一个用于预测的 GMDH 网络的步骤。

（1）数据预处理。包括数据规范化和除去数据中的静止直流成分。习惯上，对于已有的输入输出样本，在进行神经网络的训练以前，首先进行归一化处理。

（2）决定网络的输入信号数。对于预测，需要用到 n 个过去输出值。如果需要，n 的

值可以通过计算相关系数确定。

（3）将实验数据分成训练样本和测试样本。

（4）建立输入神经元层。神经元数与输入信号数 i 有关。对每个输入信号，都有一个神经元与之对应。因此，相应的神经元数为 C_i^2。

（5）将神经元权值的初始值设为 0。

（6）将训练数据组作用于输入层的每一个神经元。在 k 时刻取 y_{k-1}（$k=1,2,\cdots$）作为输入信号，y_k 为期望输出，计算每一神经元的输出误差，并修正其权值和均方误差和，当均方差和大于上一循环计算值时，训练停止。

（7）输入选择数据，计算每一神经元的输出均方差。根据差值确定一个阈值，选择方差小于阈值的神经元作为下一层神经元。

（8）当本层最小均方差大于前一层神经元的最小均方差或本层仅有一个神经元时，停止训练过程。如果训练是由于最小方差偏大而停止的，则将前一层神经元作为输出层，并重新整理网络；若训练是因本层仅有一个神经元而停止的，且本次方差小于前一层时，则以本层神经元作为输出层并重新整理网络。所谓重新整理就是指舍弃那些与输出神经元没有联系的神经元。

（9）利用评价数据组检查训练好的网络性能。评价数据组可以是上述样本数据和测试数据的结合，也可以是一组全新的数据。采用全新数据组可以在更广泛的基础上检查网络性能。

神经网络工具箱没有为 GMDH 网络提供有关的函数工具，因此，只有通过 MATLAB 的数学计算功能来实现以上的算法，在此就不再赘述了。

3.6 小　　结

本章主要介绍了有关前向型网络的工具箱函数。前向型网络包括感知器、BP 网络、线性网络、径向基网络和 GMDH 网络，它们被广泛地应用于函数逼近、模式分类和预测等领域，取得了不错的应用效果，BP 网络是其中的典型代表。在利用前向型网络解决实际问题的过程中，需要注意以下几个问题：

（1）根据实际问题选择合适的网络形式，虽然 BP 网络的应用空间比较大，但是在一些实际问题中，它的精度可能不及径向基网络。而感知器网络和线性网络只能对线性可分的模式进分类。GMDH 网络在预测方面有着不错的应用效果，但是它的训练过程比较复杂，而且没有对应的神经网络工具箱函数，限制了它的应用。

（2）三层的 BP 网络可以逼近任意的非线性映射，因此，在实际应用中，如无必要，一般采用三层的 BP 网络就足够了。BP 网络输入层和输出层的神经元个数是根据实际的输入/输出向量确定的，而隐层神经元数目的确定一直没有通用的方法，一般是首先根据工程经验确定一个大致的范围，然后通过不断实验来选定。

（3）网络结构确定后，还需要选择合理的训练函数对网络进行训练。为了避免在训练过程中陷入“局部极小”，一般应采用变速率或自适应速率的训练函数。

下一章将介绍反馈型神经网络理论及 MATLAB 实现。

第4章　反馈型神经网络理论及MATLAB实现

反馈型神经网络又称为递归网络或回归网络，它是一种反馈动力学系统，比前向神经网络具有更强的计算能力。在此类网络中，稳定性与其联想记忆的能力密切相关，因此，稳定性是一个重要的研究方向。而在前向神经网络中，注重学习方面的研究而较少关心稳定性，例如 BP 网络就是这样的。反馈神经网络注重全局稳定性方面的研究，如 Hopfield 网络在很大程度上提高了网络的稳定性。

本章包括以下内容：

- Elman 神经网络及 MATLAB 实现
- Hopfield 神经网络及 MATLAB 实现
- CG 网络模型及 MATLAB 实现
- 盒中脑（BSB）模型及 MATLAB 实现
- 双向联想记忆（BAM）及 MATLAB 实现
- 回归 BP 网络及应用
- Boltzmann 机网络及仿真

4.1　Elman 神经网络及应用

Elman 神经网络是 Elman 于 1990 年提出的，该模型在前馈网络的隐含层中增加一个承接层，作为一步延时算子，达到记忆的目的，从而使系统具有适应时变特性的能力，能直接反映动态过程系统的特性。

4.1.1　Elman 神经网络结构

Elman 型回归神经元网络一般分为 4 层：输入层、中间层（隐含层）、承接层和输出层，如图 4-1 所示。其输入层、隐含层和输出层的连接类似于前馈网络，输入层的单元仅起信号传输作用，输出层单元起线性加权作用。隐含层单元的传递函数可采用线性或非线性函数，承接层又称为上下文层或状态层，它用来记忆隐含层单元前一时刻的输出值，可以认为是一个一步延时算子。

Elman 型回归神经元网络的特点是隐含层的输出通过承接层的延迟与存储，自联到隐含层的输入，这种自联方式使其对历史状态的数据具有敏感性，内部反馈网络的加入增加了网络本身处理动态信息的能力，从而达到了动态建模的目的。

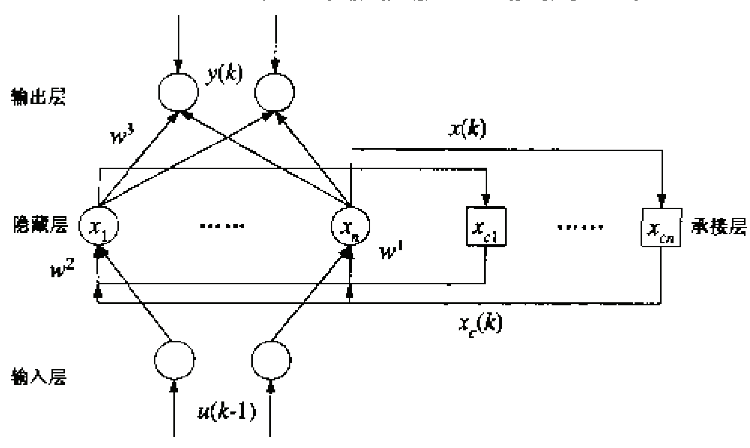


图 4-1　Elman 神经网络的模型

4.1.2　Elman 神经网络的学习过程

以图 4-1 为例，Elman 网络的非线性状态空间表达式为：

$$y(k)=g\left(w^3x(k)\right)$$
$$x(k)=f\left(w^1x_c(k)+w^2\left(u(k-1)\right)\right)$$
$$x_c(k)=x(k-1)$$

其中，y,x,u,x_c 分别表示 m 维输出结点向量，n 维中间层结点单元向量，r 维输入向量和 n 维反馈状态向量。w^3,w^2,w^1 分别表示中间层到输出层、输入层到中间层、承接层到中间层的连接权值。$g(\cdot)$ 为输出神经元的传递函数，是中间层输出的线性组合。$f(\cdot)$ 为中间层神经元的传递函数，常采用 S 函数。

Elman 网络也采用 BP 算法进行权值修正，学习指标函数采用误差平方和函数：

$$E(w)=\sum_{k=1}^{n}\left[y_k(w)-\tilde{y}_k(w)\right]^2$$

其中 $\tilde{y}_k(w)$ 为目标输出向量。

4.1.3　Elman 神经网络的工程应用

1. 工程概述

在信号处理领域，幅值检波问题实质上就是利用空间模式来对时间模式进行识别与分离。基于神经网络进行幅值检波需要一个连续时间的波形作为网络输入，网络的输出是信

号波的幅值。

2. Elman 网络的设计

下面的代码定义了两个正弦波信号 p1 和 p2 作为 Elman 网络的输入，其中 p1 的幅值为 1，p2 的幅值为 2。信号波形如图 4-2 所示。

```
t=1:20;
p1 = sin(t);
p2 = sin(t)*2;
plot(t,p1,'r');
hold on
plot(t,p2,'b--');
hold on
```

以下的代码可以产生两组向量 t1 和 t2，分别为这两个波形幅值，作为 Elman 网络的目标输出向量。

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
```

p1 和 p2 可以联合起来建立一个新的输入向量序列 p；t1 和 t2 也可以以同样的方式联合起来建立一个新的目标向量 t。这两个向量序列可以用做网络的训练样本。在训练之前，需要利用 con2seq 函数将矩阵形式的训练样本转换为序列的形式。

```
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
Pseq = con2seq(p);
Tseq = con2seq(t);
```

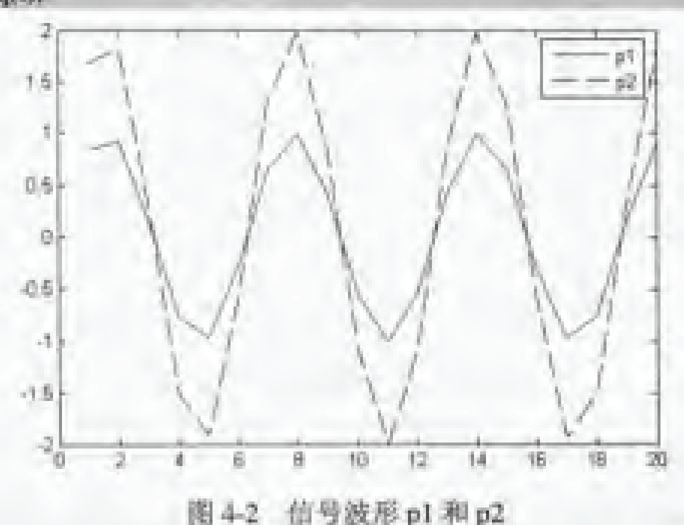


图 4-2 信号波形 p1 和 p2

利用 Elman 网络进行幅值检波就是在每一个时间步长，接受一个输入向量，输出一个向量，即幅值。因此，在训练之前的初始化过程中，输入元素的数目 R 与网络输出神经元的数目 S2 设定为：

```
R = 1;  % 输入元素的数目为 1
S2 = 1;  % 输出层的神经元个数为 1
```

理论上讲，Elman 网络中间层的神经元数目是任意选定的。但是，随着问题复杂性的不断提高，需要在中间层增加更多的神经元以使得网络的精度和速度都非常高。以下幅值

检波问题是比较简单的，所以在中间层只使用了10个神经元。

```
S1 = 10;  %中间层有10个神经元
```

接下来就可以利用newelm函数设计一个Elman神经网络。

```
net=newelm([-2,2],[S1,S2],{'tansig','purelin'});
```

上面代码表明，设计的Elman网络输入向量位于区间[-2,2]中，中间层有10个神经元，传递函数为S函数tansig，输出层有一个神经元，传递函数为线性函数purelin。训练函数采用默认值traingdx，该函数在梯度下降BP算法的基础上，在训练过程对学习速率进行自适应调整，从而提高网络的训练效率。此外，此方法在对权值更新时不仅考虑了当前的梯度方向，而且还考虑了前一时刻的梯度方向，从而降低了网络性能对参数调整的敏感性，有效地抑制了局部极小值的出现。性能函数为默认的误差平方和函数mse。

3. 网络训练

训练样本已经在上一节给出，为信号序列Pseq和Tseq。利用函数train调用traingdx对网络进行训练，训练步数设定为500次。

```
net.trainParam.epochs=500;
net=train(net,Pseq,Tseq);
```

训练结果如图4-3所示，网络经过300次训练后，网络的最小平方和误差最小，大约为0.016。因此，在接下来的测试中，采用经过300次训练的网络。

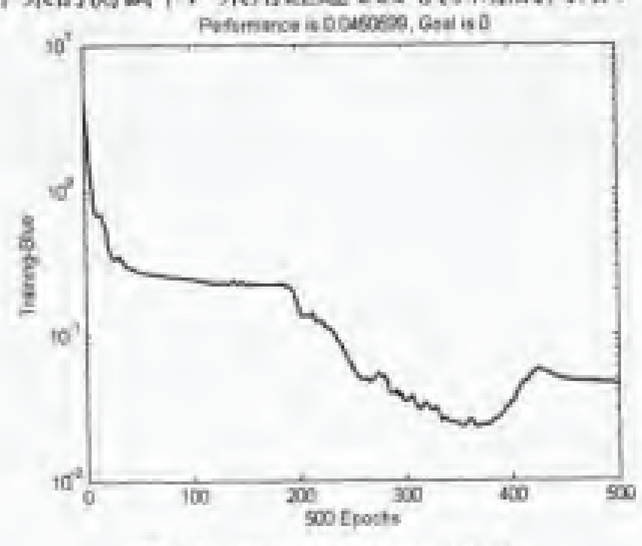


图4-3 Elman网络的训练结果

4. 网络测试与应用

训练好的网络需要利用测试数据进行测试，检验网络的预测性能是否满足要求。如果满足要求，就可以利用它来解决实际问题。为了保证测试结果的有效性和准确性，测试数据应该避免与训练数据相一致。

首先，利用训练数据对网络进行测试，看看会发生什么情况。测试代码如下：

```
Y=sim(net,Pseq);
figure;
t5=1:80;
plot(t5,cat(2,y{:}),t5,cat(2,Tseq{:}),'b--');
```

测试结果如图4-4所示。由图可见，输出信号在目标信号的两侧有小幅度的振荡，但

它随着信号幅值的变化而变化，可以只需要很少的样本就可以成功地检测出信号的幅值。

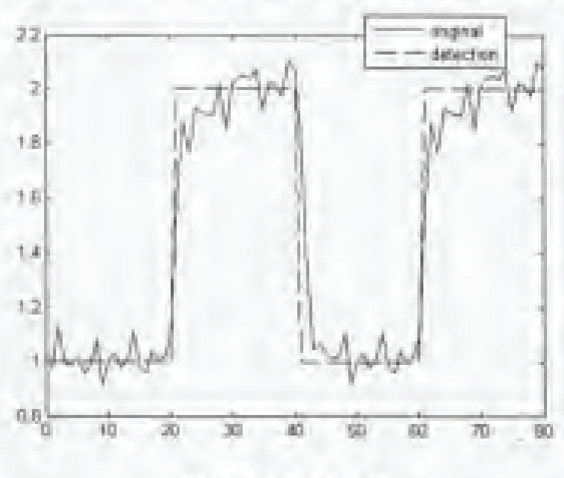


图 4-4　测试结果

细心的读者也许会问，即使网络可以很好地对训练样本进行幅值检波，但这并不意味着可以对训练样本以外的信号也可以成功地进行幅值检波。接下来就研究一下这个问题。

以下代码定义了一组新的正弦波形，并由此产生了新的测试样本，这些样本和训练样本完全不一致。

```
p3 = sin(1:20)*1.6;
t3 = ones(1,20)*1.6;
p4 = sin(1:20)*1.2;
t4 = ones(1,20)*1.2;
%产生测试输入样本 pg
pg = [p3 p4 p3 p4];
%产生测试目标样本 tg
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
```

将测试输入样本提供给网络，利用仿真函数 sim 产生网络输出，并与测试目标样本相比较，观察网络的性能，代码如下所示，测试结果如图 4-5 所示。

```
a = sim(net,pgseq);
figure;
plot(t5,cat(2,a{:}),t5,tg,'b--');
```

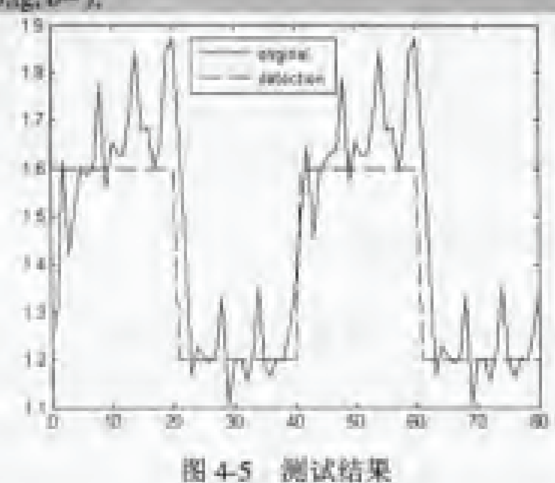


图 4-5　测试结果

由图 4-5 可见，这次网络的测试结果并不理想，幅值的检波边界非常模糊。原因有以下两点：第一，训练信号的幅值过于单调；第二，训练信号的数目太少。针对这两个原因，如果扩充训练信号的幅值，并增加训练信号的数目，网络的性能就会得到很大程度上的提高。此外，如果在中间层添加更多的神经元，并适当延长网络的训练时间，也会在一定程度上提高网络性能。

完整的 MATLAB 代码如下：

```
p1 = sin(1:20);
p2 = sin(1:20)*2;
t1 = ones(1,20);
t2 = ones(1,20)*2;
%产生训练样本 p 和 t
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
Pseq = con2seq(p);
Tseq = con2seq(t);
R = 1;  % 输入元素的数目为 1
S2 = 1;  % 输出层的神经元个数为 1
S1 = 10;  %中间层有 10 个神经元
net=newelm([-2,2],[S1,S2],{'tansig','purelin'});
%设定网络训练次数
net.trainParam.epochs=300;
net=train(net,Pseq,Tseq);
y=sim(net,Pseq);
figure;
plot(t5,cat(2,y{:}),t5,cat(2,Tseq{:}),'b--');
%利用新的信号来测试网络
p3 = sin(1:20)*1.6;
t3 = ones(1,20)*1.6;
p4 = sin(1:20)*1.2;
t4 = ones(1,20)*1.2;
%产生测试样本 pg 和 tg
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
a = sim(net,pgseq);
figure;
plot(t5,cat(2,a{:}),t5,tg,'b--');
```

4.1.4 基于 Elman 网络的空调负荷预测

空调系统逐时负荷的准确预测是实现现代控制的前提之一。但空调负荷受多种因素的影响，与诸多影响因素之间是一种多变量、强耦合、严重非线性的关系，且这种关系具有动态性，因而传统方法的预测精度不高。近年来，人工神经网络越来越引起控制理论工作者的极大兴趣。由于神经网络具有并行处理、联想记忆、分布式知识存储、鲁棒性强等特

点，尤其是它的自组织、自适应、自学习功能，从而在复杂非线性对象的辨识和控制中得到了广泛应用。

目前大多数采用的是基于 BP 算法的静态前馈神经网络。利用静态前馈网络对动态系统进行辨识，实际上是将动态时间建模问题变为静态空间建模问题。同时还需对模型结构进行定阶，特别是随系统阶次的增加或阶次未知，迅速扩大的网络结构使网络学习的收敛速度减慢，并造成网络输入结点过多、训练困难及对外部噪声敏感等弊病。相比之下，动态回归神经网络（RNN）提供了一种极具潜力的选择，它能够更生动、更直接地反映系统的动态特性，代表了神经网络建模、辨识与控制的发展方向。Elman 回归神经网络是一种典型的动态神经元网络，它是在 BP 网络基本结构的基础上，通过存储内部状态使其具备映射动态特征的功能，从而使系统具有适应时变特性的能力。因此，考虑到空调系统具有动态性的特点，尝试采用 Elman 网络进行预测。

1. 样本设计

空调系统的逐时负荷是按时间顺序排列的数字序列，它们之间具有某种统计意义上的关系，这种关系很难用确定的函数或方程组来描述。

基于神经网络对时间序列进行预测，通常是根据已有的样本数据对网络进行训练。如果希望用过去的 N（$N \geqslant 1$）个数据预测未来 M（$M \geqslant 1$）个时刻的值，即进行 M 步预测，可取 N 个相邻的样本为滑动窗，并将它们映射为 M 个值，这 M 个值代表在该窗之后的 M 个时刻上的样本的预测值。如表 4-1 所示，列出了样本数据的一种分段方法，该表把训练数据分成 K 段长度为（$N+M$）的有一定重叠的数据段，每一段的前 N 个数据作为网络的输入，后 M 个数据作为网络的输出。

表 4-1 样本数据的分段方法

N 个输入	M 个输出
$X_1,X_1,\cdots,X_N$	$X_{N+1},X_{N+2},\cdots,X_{N+M}$
$X_2,X_3,\cdots,X_{N+1}$	$X_{N+2},X_{N+3},\cdots,X_{N+M+1}$
……	……
$X_K,X_{K+1},\cdots,X_{N+K-1}$	$X_{N+K},X_{N+K+1},\cdots,X_{N+M+K-1}$

在实际的空调系统中，i 时刻的负荷不仅受到 i 时刻外部环境及内部因素的影响，由于负荷形成的时滞性，而且要受到 $i-1,i-2,\cdots,i-n$ 等时刻诸多因素的影响，具有动态性。这里对空调负荷的预报是以日为单位的，即利用前 n 日的负荷数据直接预测第 $n+1$ 日的 1 个逐时负荷。

一般来说，办公室空调系统的负荷高峰通常出现在每天的 9～19 时之间，出于篇幅的原因，这里只对每天上午的逐时负荷进行预测，即预测每天 9～12 时共 4 个小时的负荷。如表 4-2 所示，给出了从 2003 年 7 月 1 日至 7 日的每天上午的空调负荷数据，其中所有的数据都已经归一化处理过了。

表 4-2 空调负荷数据

时 间	负荷数据
2003-7-1	0.4413 0.4707 0.6953 0.8133
2003-7-2	0.4379 0.4677 0.6981 0.8002
2003-7-3	0.4517 0.4725 0.7006 0.8201
2003-7-4	0.4557 0.4790 0.7019 0.8211
2003-7-5	0.4601 0.4811 0.7101 0.8298
2003-7-6	0.4612 0.4845 0.7188 0.8312
2003-7-7	0.4615 0.4891 0.7201 0.8330

现在利用前 6 天的数据作为网络的训练样本，每 3 天的负荷作为输入向量，第 4 天的负荷作为目标向量。这样可得到 3 组训练样本。第 7 天的数据作为网络的测试样本，主要看网络能否合理地预测出当天的数据。



对于空调负荷预报来说，只考虑历史数据是不够的。对于一个实际的时间序列，它的预测值不仅取决于历史数据，而且受许多突变因素的影响。为此，在上述模型的基础上，加上控制变量以提高精度。空调系统的负荷除与历史运行数据有关以外，还受到预测日的气象参数（温度、湿度）及预测日的特性的影响。由于工作日和节假日的负荷不同，因此还得考虑时间特征值。这里由于篇幅的原因，对预报模型简单化，但这并不影响 Elman 预测功能的演示。

2. 网络的设计与训练

合理确定 Elman 网络的结构是预测性能的基础。实际上结构的确定尤其是中间层神经元数目的确定是一个经验性的问题，还需要大量的实验。由样本数据可知，网络输入层应该有 12 个神经元，输出层应该有 4 个神经元。根据经验，可以先将中间层神经元的数目设置为 13 个。代码如下：

```
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newelm(threshold,[13,4],{'tansig','purelin'});
```

以上代码创建了一个 Elman 网络，threshold 规定了网络输入元素的最大值和最小值；[13,4]和{'tansig','purelin'}表示网络中间层神经元数目为 13，传递函数为 tansig，输出层神经元数目为 4，传递函数为 purelin。

接下来对网络进行训练，代码如下：

```
net.trainParam.epochs=1000;
net=train(net,P,T);
```

训练之前只设定了训练步数，训练目标误差为默认值 0。P 为训练样本中的输入向量，T 为目标向量。结果如下：

```
......
TRAINGDX, Epoch 975/1000, MSE 0.000273331/0, Gradient 0.00367873/1e-006
TRAINGDX, Epoch 1000/1000, MSE 0.000270021/0, Gradient 0.00101137/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.
```

可见经过 1000 次训练后，网络的训练误差为 0.00027 左右。训练结果如图 4-6 所示，

这已经是比较小的数据了，我们认为可以接受。

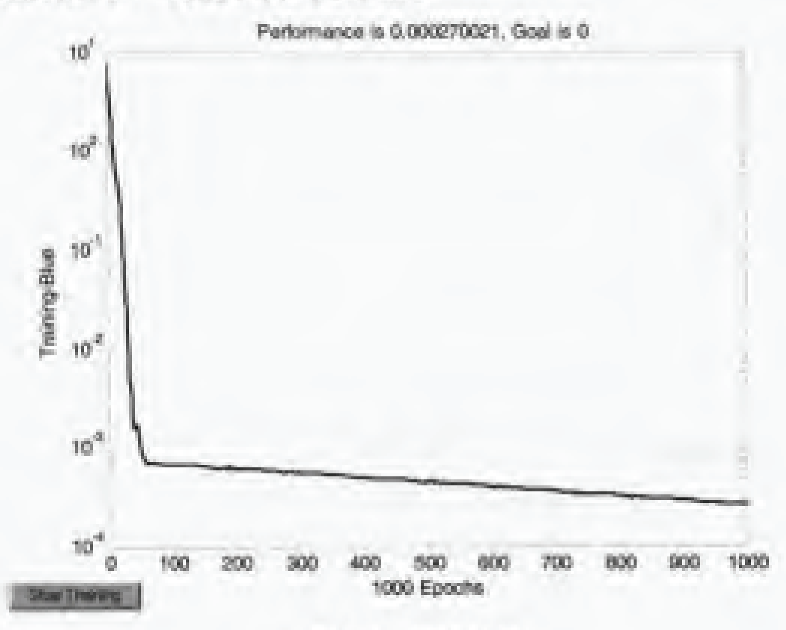


图 4-6　训练结果

然后对网络进行测试，看网络能否精确地预测出 7 月 7 日的数据，代码为：

```
y=sim(net,p_test)
```

结果为：

```
y=
0.4559
0.4189
0.7276
0.8332
```

可见，网络的预报误差是比较小的，但是，10 时的预报值出现了一个相对较大的误差，如图 4-7 所示。这是因为训练样本太小造成的。



图 4-7　误差曲线

为了检验中间层神经元数目对于网络预测性能的影响，将其分别设定为 17 次和 23 次后观察其训练曲线和预测误差曲线，如图 4-8、图 4-9 和图 4-10 所示。

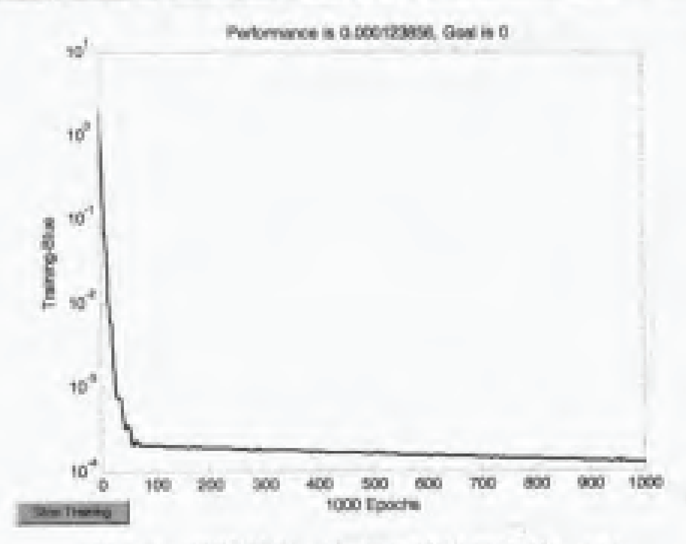


图 4-8　训练结果（中间层神经元个数：17）

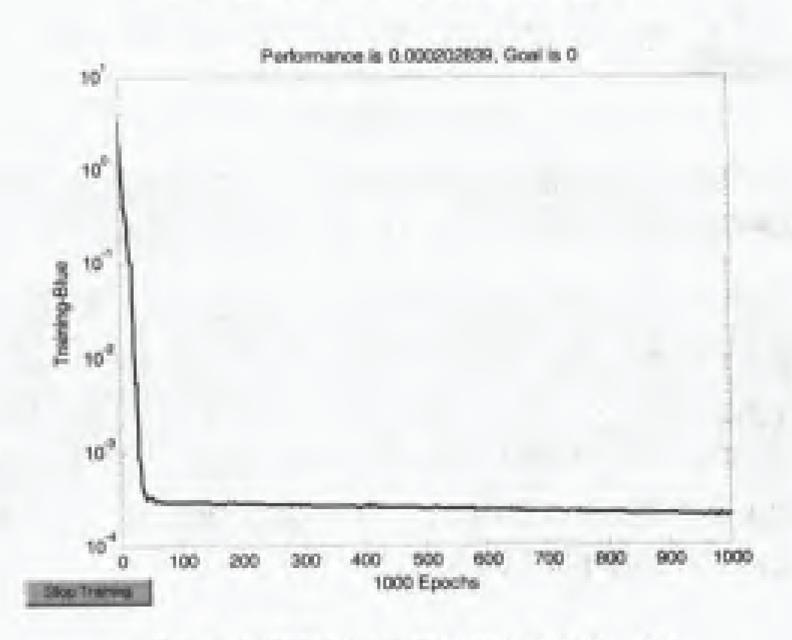


图 4-9　训练结果（中间层神经元个数：23）

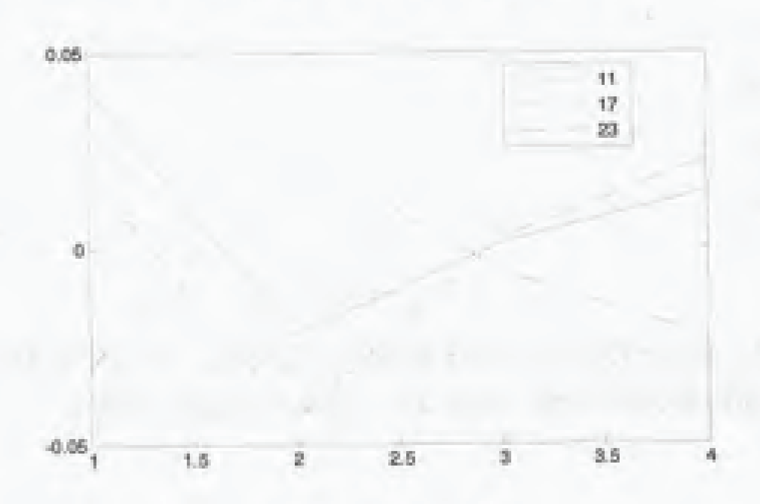


图 4-10　误差曲线

可知，中间层神经元为 17 个时，网络的预测误差最小，也就是预测性能最好。因此，对于本例，中间层神经元的最佳数目应该是 17 个。

本实例的完整 MATLAB 代码为：

```
P=[0.4413 0.4707 0.6953 0.8133 0.4379 0.4677 0.6981 0.8002 0.4517 0.4725 0.7006 0.8201;
   0.4379 0.4677 0.6981 0.8002 0.4517 0.4725 0.7006 0.8201 0.4557 0.4790 0.7019 0.8211;
   0.4517 0.4725 0.7006 0.8201 0.4557 0.4790 0.7019 0.8211 0.4601 0.4811 0.7101 0.8298;]'
T=[0.4557 0.4790 0.7019 0.8211;
   0.4601 0.4811 0.7101 0.8298;
   0.4612 0.4845 0.7188 0.8312]';
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
a=[11 17 23];
for i=1:3
    net=newelm(threshold,[a(i),4],{'tansig','purelin'});
    net.trainParam.epochs=1000;
    net=init(net);
    net=train(net,P,T);
    y=sim(net,p_test);
    error(i,:)=y'-t;
end
hold off;
plot(1:4,error(1,:));
hold on;
plot(1:4,error(2,:),'-.');
hold on;
plot(1:4,error(3,:),'--');
hold off;
```

4.2 Hopfield 神经网络及 MATLAB 实现

Hopfield 网络作为一种全连接型的神经网络，曾经为人工神经网络的发展进程开辟了新的研究途径。它利用与阶层型神经网络不同的结构特征和学习方法，模拟生物神经网络的记忆机理，获得了令人满意的结果。这一网络及学习算法最初是由美国物理学家 J.J Hopfield 于 1982 年首先提出的，故称为 Hopfield 网络。1985 年 Hopfield 本人和 Tank 用这种网络模型成功解决了优化组合问题中的具有典型意义的旅行商问题 TSP，在所有随机选择的路径中找到了十万分之一的最优路径，这是当时神经网络研究工作中具有突破性的进展。

Hopfield 网络有离散型和连续型两种形式，这里只介绍离散型 Hopfield 网络，因为离散型 Hopfield 网络的结构比较简单，在实际工程中的应用比较广泛。如无特别声明，以后凡出现 Hopfield 网络，都是指离散型的。

4.2.1 Hopfield 网络描述

Hopfield 网络结构如图 4-11 所示，它是一种单层反馈性非线性网络，每一个结点的输出均反馈到其他结点的输入，整个网络都不存在自反馈。

J.J Hopfield 利用模拟电路（电阻、电容和运算放大器）实现了对网络的结点（神经元）的描述，如图 4-12 所示。

假设网络共有 n 个这样的神经元组成，可得出

$$\frac{dx_i}{dt}=-\frac{1}{\tau}x_i+\frac{1}{C_i}\sum_j w_{ij}y_j+\theta_j$$
$$y_i=f(x_i)$$

其中，$x=u$，$y=V$，$\tau=R_i'C_i$，$\theta=I/C$，$w_{ij}=1/R_{ij}$，$w_{ii}=0$，$w_{ij}=w_{ji}$，且 $\frac{1}{R_i'}=\frac{1}{R_i}+\sum_j\frac{1}{R_{ij}}$。

传递函数 $f(x)$ 为 S 函数。

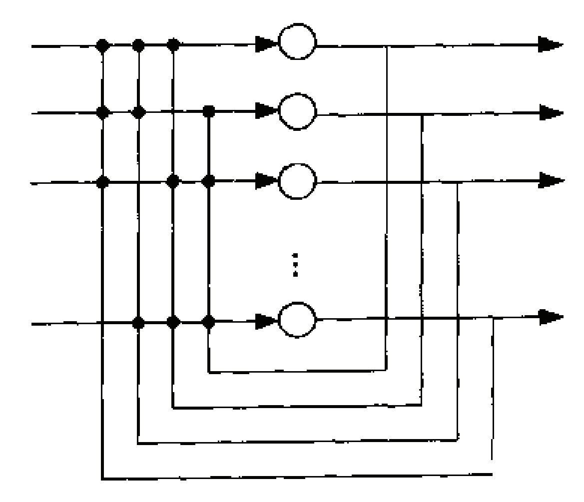


图 4-11 Hopfield 网络结构

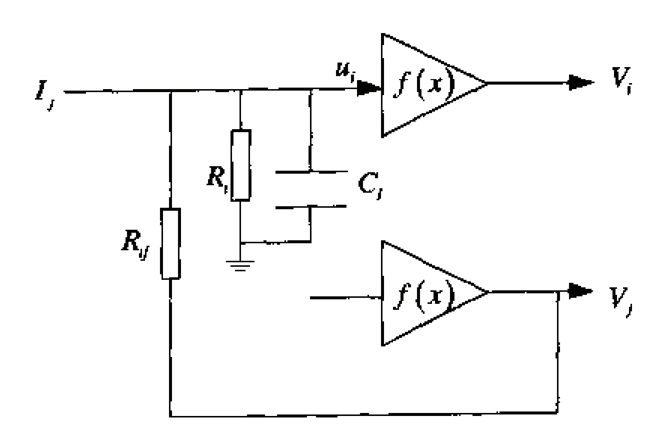


图 4-12 Hopfield 神经元的模拟电路

由此可见，R_i,C_i 的并联模拟了生物神经元的时间常数，w_{ij} 模拟了神经元间的突触特

性即权值，运算放大器模拟了神经元的非线性特性，偏置电流 I_j 相当于阈值。

Hopfield 网络是渐近稳定的，随着时间的推移，网络状态向能量减小的方向移动，稳定平衡状态就是能量的极小点。因此，如果网络的初始状态在稳定平衡状态，则其状态不变；否则，网络需要运动到稳定平衡状态。

4.2.2 Hopfield 网络的学习过程

网络的学习过程实际上就是权值调整过程，Hopfield 网络的学习目的就是调整连接权值，以使得网络的稳定平衡状态就是所要求的状态。

Hopfield 网络常采用的学习算法是 Hebb 学习规则，即权值调整规则为：若第 i 个和第 j 个神经元同时处于兴奋状态，那么它们之间的连接应该增强，权值增大。

$$\triangle w_{ij}=ay_iy_j \qquad a>0$$

假设要求网络要有 p 个正交稳态 $V^s=\left(V_1^s,V_2^s,\cdots,V_n^s\right)$， $s=1,2,\cdots,p$，则

$$w_{ij}=\sum_{s=1}^{p}V_i^sV_j^s$$

若增加新的稳态 V^{p+1}，则

$$w_{ij}=w_{ij}+V_j^{p+1}V_j^{p+1}$$

4.2.3 几个重要结论

（1）联想记忆功能。由于网络可以收敛于稳定状态，因此可用于联想记忆。若将稳态视为一个记忆，则由初始状态向稳态收敛的过程就是寻找记忆的过程，初态可认为是给定的部分信息，收敛过程可认为是从部分信息找到了全部信息，则实现了联想记忆的功能。联想记忆的一个重要特性是由噪声输入模式反映出训练模式。

（2）优化计算。若将稳态视为某一优化问题目标函数的极小点，则由初态向稳态收敛的过程就是优化计算过程。

（3）网络渐近稳定的前提是 $w_{ij}=w_{ji}$。

（4）网络的应用。Hopfield 网络多用于在控制系统的设计中求解约束优化问题，另外在系统辨识中也有应用。

4.2.4 Hopfield 网络的 MATLAB 开发

本节利用 MATLAB 中的 Demos 功能，演示如何利用神经网络工具箱创建一个两神经元的 Hopfield 网络。

1. Demos 功能的启动

在命令行窗口中键入 Demos 后按回车键，出现如图 4-13 所示的 Demos 标签页。首先在左侧的目录树中选择 Neural Network→Hopfield Networks→Hopfield two neuron design 节点，然后单击窗口右上角的 Run this demo 链接。

第 1 步：单击 Run this demo 链接后，出现两神经元 Hopfield 网络 Demos 的初始界面，如图 4-14 所示。可以看出，本次演示共有 8 步。这是第 1 步。

第 2 步：单击图 4-14 所示窗口中的【Start>>】按钮，出现如图 4-15 所示的窗口。本窗口中定义了网络的目标向量 T=[1 –1;-1 1]，向量 T 的两列也是存储在网络中的目标稳定点。然后，单击【Next>>】按钮，进入第 3 步。

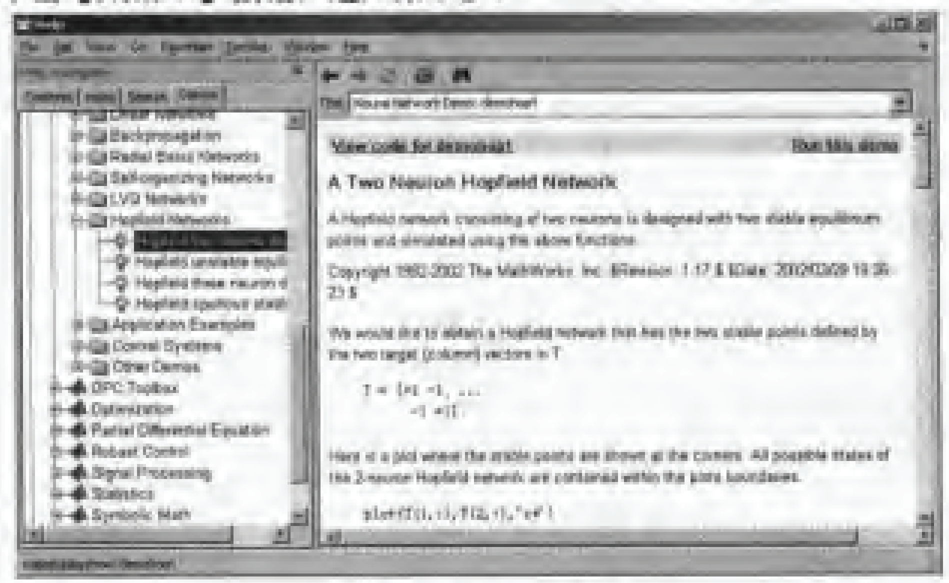


图 4-13　Demos 标签页

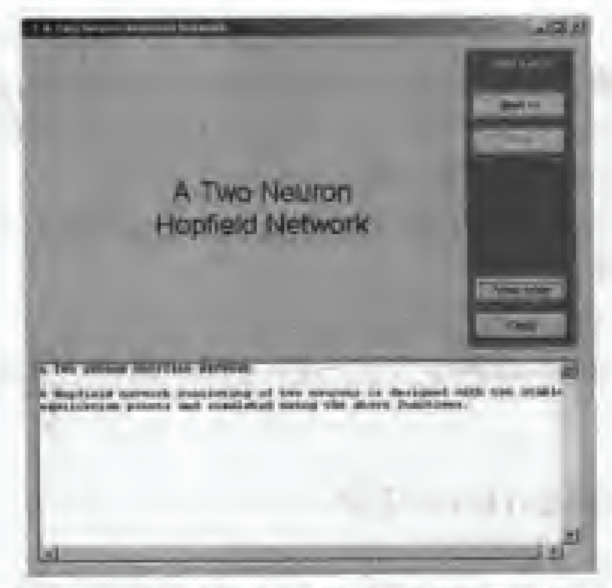


图 4-14　两神经元 Hopfield 网络 Demos 初始界面



图 4-15　目标向量 T 定义窗口

第 3 步：该步骤绘制 Hopfield 网络状态空间，其中用星号标识了上述的两个稳定点。如图 4-16 所示，图中的上半部分为绘制的稳定点位置，下半部分为用于绘制的代码。单击【Next>>】按钮，进入第 4 步。



图 4-16　绘制稳定点的位置

```
plot(T(1,:),T(2,:),'r*')
axis([-1.1 1.1 -1.1 1.1])
title('Hopfield Network State Space')
xlabel('a(1)');
ylabel('a(2)');
```

第 4 步：这一步创建了一个有两个神经元的 Hopfield 网络，其对应的窗口和图 4-16 基本相同，惟一的区别就是下半部分的代码窗口，其中的代码为：

```
net = newhop(T);
```

单击【Next>>】按钮，进入第5步。

第5步：在该步中，使用目标向量T对网络进行仿真，检查T是否已经存储到网络中。其对应的窗口和图4-16基本相同，惟一的区别也是下半部分的代码窗口，代码为：

```
[Y,Pf,Af] = sim(net,2,[],T);
Y;
```

输出结果为：

```
Y=
   1   -1
  -1    1
```

结果和设想的一致，网络的输出就是目标向量T。单击【Next>>】按钮，进入第6步。

第6步：该步用于测试网络的稳定性。测试方法为：给网络提供一个随机向量，并且对网络做20次仿真，检查输出结果。设想输出值应该为网络的某个稳定点。该步对应的窗口和图4-16几乎一致，区别在于代码的不同。代码为：

```
%创建一个随机列向量
a = {rands(2,1)};
%对网络进行20次仿真
[y,Pf,Af] = sim(net,{1 20},{},a);
```

这一步中还没有显示输出结果。单击【Next>>】按钮，进入第7步。

第7步：该步绘制了上一步给定的随机点的运行轨迹，结果如图4-17所示。可以看出，仿真的输出结果最终达到了稳定点（-1,1），这和设想的情况是吻合的。为了更有效地说明Hopfield网络对目标点的跟踪能力，还需要做进一步的测试。单击【Next>>】按钮，进入最后一步。

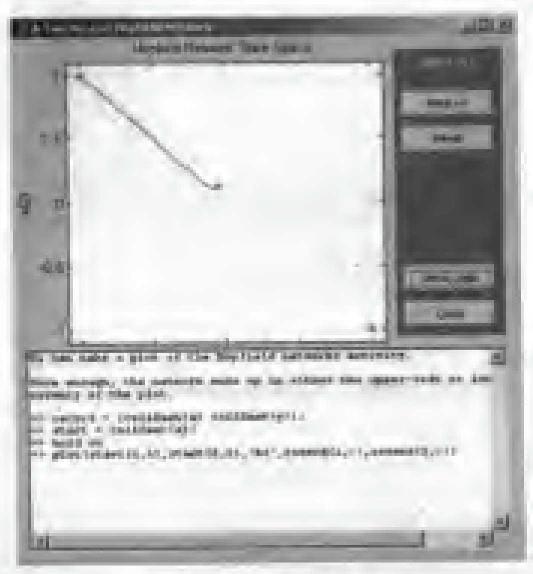


图4-17　仿真结果

第8步：在这一步中，选择了25个随机值作为起始点，看在仿真后的输出能否回到稳定点。该步的代码为：

```
color = 'rgbmy';
for i=1:25
    a = {rands(2,1)};
```

```
        [y,Pf,Af] = sim(net,{1 20},{},a);
        record=[cell2mat(a) cell2mat(y)];
        start=cell2mat(a);
        plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:),color(rem(i,5)+1))
    end
```

该步仿真的结果如图4-18所示。可见，所有的仿真输出都回到了相应的稳定点。起始点离哪一个稳定点近，就最终趋向于哪一个稳定点。正是由于具有这种对一个初始输入能够找到相应的最近的存储值的能力，使得Hopfield网络十分有用。



图4-18 25个初始条件下的仿真结果

4.2.5 基于Hopfield网络的数字识别

字符识别是计算机模式识别的一个重要方面。作为字符识别的组成部分之一的数组识别在邮政、交通及商业票据管理方面有着极高的应用价值。目前有很多种方法用于字符识别，主要分为神经网络识别、概率统计识别和模糊识别等。近年来，神经网络识别技术发展较快，由于其本身具有自学习和自组织等优良特性而日益受到重视。目前，对印刷体的字符识别技术已经比较成熟了，而手写体字符由于书写者的因素，使得字符图像的随意性很大，比如笔画的粗细、字体的大小、手写体的倾斜度和局部扭曲等直接影响到字符的正确识别。因此，手写体字符的识别是字符识别领域最有挑战性的课题。

出于篇幅的原因，这里不研究手写体字符的识别，而是针对印刷体的数字识别来设计一个Hopfield网络。正常的和被噪声污染的数字如图4-19所示。

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

图4-19 正常的数字和被污染的数字

例 4.1 设计一个 Hopfield 网络，使其具有联想记忆功能，能正确识别阿拉伯数字，当数字被噪声污染后仍可以正确地识别。

假设网络由 10 个初始稳态值 0~9 构成，即可以记忆 10 种数字。每个稳态由 10×10 的矩阵构成，该矩阵用于模拟阿拉伯数字点阵。所谓数字点阵，就是将数字化分成很多小方块，每一个小方块都对应着一部分数字。这里将数字划分成一个 10×10 方阵，其中，有数字的方块用 1 表示，空白处用 – 1 表示，如图 4-20 所示。

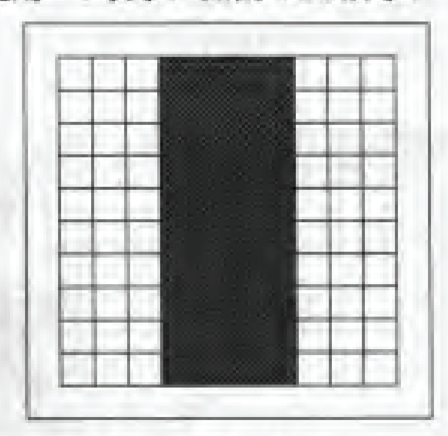


图 4-20　数字 1 的数字点阵

这样一来，就有：

```
one=[-1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1
    -1 -1 -1 -1 -1 1 1 1 1 -1 -1 –1   -1 -1 -1 1 1 1 1 -1 -1 –1   -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1
    1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1 -1 -1];
two=[1 1 1 1 1 1 1 1 -1 -1   1 1 1 1 1 1 1 1 -1 –1   -1 -1 -1 -1 -1 -1 1 1 -1 –1   -1 -1 -1 -1 -1 -1 1 1 -1
-1   1 1 1 1 1 1 1 1 -1 –1   1 1 1 1 1 1 1 1 -1 -1   1 1 -1 -1   -1 -1 -1 -1 -1 -1 1 1 -1 -1   -1 -1 -1 -1 –1
-1 1 1 1 1 1 1 1 1 -1 –1   1 1 1 1 1 1 1 1 -1 -1];
```

出于篇幅的原因，这里只列出 1 和 2 的点阵表示形式。利用这两个向量构成一个训练样本，并由此构建一个 Hopfield 网络。

```
T=[one;two]';
uet=newhop(T);
```

给出一个受到噪声污染的数字 2 的点阵 No2：

```
No2={[1 1 1 -1 1 1 -1 1 -1 -1   1 1 1 1 1 1 1 1 -1 -1   -1 1 -1 1 -1 -1 1 1 -1 -1   -1 -1 1 -1 1 -1 1 1 1 -1
-1 1 1 1 1 1 1 1 1 -1 –1   1 1 1 1 1 1 1 1 -1 -1   1 1 -1 -1 -1 -1 -1 -1 -1 –1   1 1 -1 -1 -1 -1 -1 -1 -1 –1
1 1 1 -1 1 1 1 1 -1 -1 1 1 -1 1 1 1 1 1 1 -1 -1]'};
```



所谓噪声，实际上就是数字点阵中的某些位由原来的 1 时变为 – 1，或者相反。如图 4-21 所示，就是受到污染后数字 1 的数字点阵。

接下来尝试利用刚刚创建的 Hopfield 网络将受到噪声污染的数字 2 识别出来。

```
tu2=sim(net,{1,5},{},no2);
tu2{3}'
```

结果为：

ans=1 1 1 1 1 1 1 1 -1 –1   1 1 1 1 1 1 1 1 -1 –1   -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1
    1 1 -1 -1   1 1 1 1 1 1 1 1 -1 -1   1 1 1 1 1 1 1 1 -1 -1   1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1

-1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 1 1 1 -1 -1　1 1 1 1 1 1 1 1 -1 –1。

结果和数字 2 的正常点阵是一致的，说明网络从受到污染的数字 2 的点阵中识别出了数字 2，此网络是有效的。

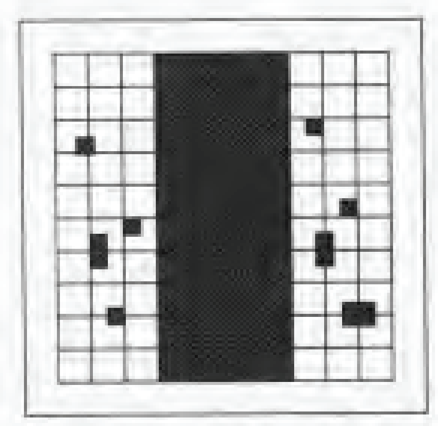


图 4-21　受噪声污染的 1 的数字点阵

本实例的 MATLAB 代码为：

```
one=[-1 -1 -1 1 1 1 1 -1 -1 -1 –1   -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1
    -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1   -1 -1 -1 1 1 1 1 -1 -1 –1   -1 -1 -1 1 1 1 1 -1 -1 –1   -1 -1 -1 1 1 1
    1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 –1   -1 -1 -1 1 1 1 1 -1 -1 -1];
two=[1 1 1 1 1 1 1 1 -1 -1   1 1 1 1 1 1 1 1 -1 –1   -1 -1 -1 -1 -1 -1 1 1 -1 -1   -1 -1 -1 -1 -1 -1 1 1 -1
    -1   1 1 1 1 1 1 1 1 -1 -1   1 1 1 1 1 1 1 1 -1 –1   1 1 -1 -1   -1 -1 -1 -1 -1 -1 1 1 -1 -1   -1 -1 -1
    -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1];
T=[one;two]';
net=newhop(T);
no2={[1 1 1 -1 1 1 -1 1 -1 -1   1 1 1 1 1 1 1 1 -1 -1   -1 1 -1 1 -1 -1 1 1 -1 -1   -1 -1 1 -1 1 1 -1 1 1 -1 -1
    1 1 1 1 1 1 1 1 -1 -1   1 1 1 1 1 1 1 1 -1 -1   1 1 -1 -1 -1 -1 -1 -1 -1 -1   1 1 -1 -1 -1 -1 -1 -1 -1 -1
    1 1 1 -1 1 1 1 1 -1 -1   1 1 -1 1 1 1 1 1 -1 -1]'};
tu2=sim(net,{1,5},{},no2);
tu2{3}'
```

4.3　CG 网络模型及应用

CG 网络模型是由 Cohen 和 Grossberg 提出的，它是 Hopfield 神经网络的一种推广形式。和 Hopfield 网络不同，CG 网络神经元的状态不再是二值的，而是根据一组常微分方程连续地变化。CG 网络主要用于联想记忆，信息存在于系统的局部最小值上。CG 模型是综合了神经生物学、群体生物学和进化论的观点而提出的，在信号处理和优化问题等领域有着广泛的应用。

4.3.1　CG 神经网络理论

CG 网络模型可用如下的常微分方程进行描述：

$$x_i' = -a_i(x_i)\left(b_i(x_i) - \sum_{j=1}^{n} t_i s_j(x_j)\right) \qquad i = 1, 2, \ldots, n$$

其中，n 为大于等于 2 的数，表示网络中的神经元数目；x_i 表示第 i 个神经元对应的状态；a_i 为正调函数，即 $a_i \ge 0$；≥为单调递增函数，即 $b_i' \ge 0$；|≥t_{ij} 构成的 $n \times n$ 维矩阵 T 为连接权值矩阵，描述了各神经元之间的耦合程度，且 $t_{ij} = t_{ji}$；s_j 为激励函数，$\sum_{j=1}^{n} t_i s_j\left(x_j\right)$ 表示神经元的输入。

上式有一个重要的特点，即全局收敛性，也就是说，如果权矩阵 T 是对称的，在任何给定的初始条件下，上式总能收敛于系统的某一个等价形式。

4.3.2　基于 CG 网络的有限元分析

CG 网络在结构分析中的应用比较广泛，下面简单介绍一下基于 CG 网络的有限元分析的步骤。

（1）有限元预处理，包括有限元网络划分，建立单位刚度矩阵和结点载荷矩阵等。

（2）形成神经网络计算系统，选择合适的参数，组织神经网络求解动力学方程。

（3）求解神经网络动力学方程。该步骤中最关键的问题是得到方程的稳态解，一般采用数值方法来模拟整个动力学过程。常用的数值方法有最速下降法、共轭梯度法和龙格—库塔法等。

由于神经网络工具箱没有为 CG 网络设计专门的函数，因此，在确定了动力学方程后，需要利用 MATLAB 强大的数学计算及分析功能实现基于 CG 网络的有限元分析，具体实例在此不再列举。

4.4　盒中脑（BSB）模型及 MATLAB 实现

盒中脑（Brain-State-in-a-Box，BSB）神经网络模型首先是由 Anderson 等人于 1977 年提出的，Golden 等人对该模型进行了深入的研究。BSB 模型是一种结点之间存在横向连接和结点自反馈的单层网络，可用做自联想最邻近分类器，并可存储任何模拟向量模式。

4.4.1　BSB 神经网络模型描述

BSB 网络模型可用如下方程描述：

$$X(k+1) = g\left(X(k) + \alpha W X(k)\right) \qquad k = 0, 1, 2, \cdots$$

初始条件为 $X(0)=X_0$，其中 $X(k)=(x_1(k),x_2(k),\cdots,x_n(k))\in \mathrm{R}^n$ 表示 k 时刻的状态向量，参数 α 是一正值，用于控制层内反馈的大小。$W\in \mathrm{R}^{n\times n}$ 为对称的权矩阵，传递函数 g 的第 i 个坐标通常为以下形式：

$$g_i(X)=\begin{cases}1 & x_i\geqslant 1\\ x_i & |x_i|<1\\ -1 & x_i\leqslant -1\end{cases}$$

随着时间的推移，每个状态 x_i 逐渐趋近于 ±1。实际上，当系统达到某个平衡态后，状态 $(x_1,x_2,\cdots,x_n)$ 进入由 $(\pm1,\pm1,\cdots,\pm1)$ 构成的盒子的某一角。

4.4.2 BSB 的 MATLAB 实现

神经网络工具箱中没有为 BSB 网络提供专门的函数工具。因此，无法利用神经网络工具箱中的函数创建、训练并应用网络。但是，Hugh Pasika 于 1997 年基于 MATLAB 平台开发了 BSB 网络的实现函数。代码如下：

```
function c=bsb(x,beta,multi)

% function c=bsb(x,beta)
%
%   This m-file duplicates the Brain State in a Box Experiment.
%       x        - input vector
%       beta     - feedback factor
%       c        - number of iterations required for convergence
%
% Hugh Pasika 1997

hold on
flag=0;    x=x(:);    c=2;   %c is a general purpose counter
W=[.035 -.005; -.005 .035];

set(gca,'YLim',[-1 1]);          set(gca,'XLim',[-1 1])   % set axes

plot(x(1),x(2),'ob')             % plot first point
orig=x';
plot([0,0],[1,-1],'-');         plot([1,-1],[0,0],'-')        % plot center lines
set(gca,'YTick',[-1   1 ]);     set(gca,'XTick',[-1    1 ])   % label plot

while flag < 1,
y=x+beta*W*x;
```

```
x=(y(:,:) < -1 )*(-1) + (y(:,:)>1) + (y(:,:) > -1 & y(:,:) < 1).*y;
u(c,:)=x';
c=c+1;
    if u(c-1,:) == u(c-2,:),
       flag=10;
       c=c-3;
    end
end
u=u(2:c+1,:);
orig
plot([orig(1,1) u(1,1)], [orig(1,2) u(1,2)],'-b')
plot(u(:,1), u(:,2),'ob')
plot(u(:,1), u(:,2),'-b')
drawnow
fprintf(1,'It took %g iteration for a stable point to be reached.\n\n',c);
set(gca,'Box','on')
hold off
```

读者可以将上述代码保存为一个名为 bsbdefine.m 的文件，存储到 X:/MATLAB7/work 文件夹中，其中 X 为 MATLAB 的安装盘的盘符。然后在 MATLAB 主窗口中将 Current Directory 当前目标修改为 X:/MATLAB7/work，在命令行窗口中输入“help bsb”后按回车键，出现如下的帮助信息：

```
function c=bsb(x,beta)
    This m-file duplicates the Brain State in a Box Experiment.
     x        - input vector
     beta    - feedback factor
     c        - number of iterations required for convergence
Hugh Pasika 1997
```

帮助信息解释了函数各个参数的意义，其中：

- x：输入向量；
- beta：反馈因子；
- c：最大迭代次数。

下面给定输入向量和反馈因子，演示 bsb 函数的功能。令 x=[0.5;-0.6]，beta=0.5，c=100，c 用于限制迭次次数。迭代终止的充分条件是网络已经收敛或者网络的迭代次数达到最大值 c。在命令行窗口中输入以下代码后按回车键。

```
x=[0.5;-0.6];
beta=0.5;
c=100;
bsb(x,beta,c)
```

结果为：

```
orig =
      0.5000    -0.6000
It took 25 iteration for a stable point to be reached.
ans =
```

25

其中 orig 表示网络的初始输入值为（0.5,-0.6）。ans=25 表示网络经过 25 次迭代后，初始值就达到了箱子的一角（1,-1），因为初始值与它的距离最小。输入向量的收敛轨迹如图 4-22 所示。

图 4-22 输入向量的收敛轨迹

4.5 双向联想记忆（BAM）及 MATLAB 实现

双向联想记忆（Bidirectional Associative Memory，BAM）网络是由 Kosko 最早提出来的,它是一种可以记忆模式对的两层非线性反馈神经网络。它采用前向和反向双向联想，从一个输入对(A,B)回忆另一个相关的双极性向量对(A_k,B_k)，故称为双向联想存储器。BAM 神经网络在智能系统中有着比较广泛的应用。

4.5.1 Kosko 型 BAM 网络模型

BAM 网络可以存储两组向量：n 维向量 A 和 p 维向量 B。

$$A=\left[a_0,a_1,\cdots,a_{n-1}\right]^{\mathrm{T}}，其中，-1<a_i<1,i=0,1,\cdots,n-1$$

$$B=\left[b_0,b_1,\cdots,b_{p-1}\right]^{\mathrm{T}}，其中，-1<b_j<1,j=0,1,\cdots,p-1$$

以上两组向量构成一组向量对(A_s,B_s)， $s=0,1,\cdots,M-1$，共有 M 对样本向量。将它们提供给 BAM 网络即可进行由 A 到 B 或由 B 到 A 的双向联想。如果有噪声或缺损时，联想功能可使样本对复原。

BAM 网络有很多种结构形式，Kosko 型 BAM 网络是其中最基本的一种，其结构如图 4-23 所示。图中与向量 A 对应的一层有 n 个结点，另一层对应着向量 B，由 p 个结点组成。两层之间双向连接。假定由 B 到 A 的传输方向为正向，则由 A 到 B 的传输方向为反向。正向的权值矩阵为 W，反向权值矩阵为 W^{T}。

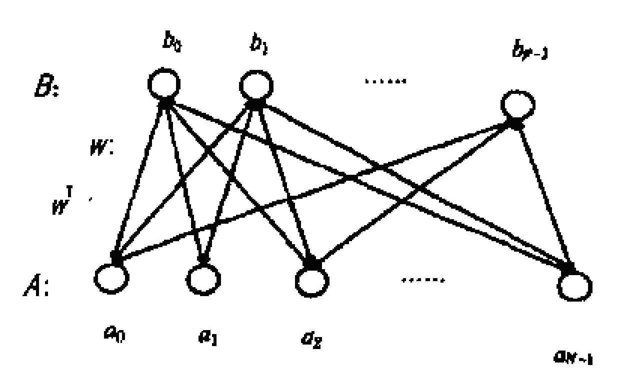


图 4-23 Kosko 型 BAM 网络结构

如果输入矢量由上层加入，且对应于网络中 B 的稳定状态，则经过权值 W 的作用产生 A 稳定状态。同理，若输入矢量由下层加入，且对应于 A 的稳定状态，则经 W^{T} 的作用产生 B 稳定状态。

对于 BAM 网络，若输入为任意矢量，网络状态转换过程如下：

$$WB(t) \to A(t+1)$$
$$W^T A(t+1) \to B(t+2)$$
$$WB(t+2) \to A(t+3)$$

直到 A 和 B 均达到稳定状态，转换过程就结束。

BAM 网络按照 Hebb 规则进行学习，在给定双极性向量对 (A_s, B_s)， $s = 0,1,\cdots,M-1$ 的前提下，权值矩阵的表达式为：

$$W = \sum_{s=0}^{M-1} A_s B_s^{\mathrm{T}}$$
$$W^{\mathrm{T}} = \sum_{s=0}^{M-1} B_s A_s^{\mathrm{T}}$$

经证明，网络状态经过一定的变换过程后，网络将会进入某一个稳定状态。

4.5.2 BAM 网络的实例分析

利用神经网络实现联想功能，一般有两种联想形式，即自联想和异联想（双向联想）。Hopfield 网络属于自联想模式，而 BAM 网络属于双向联想模式，这种网络的特点是会收敛于吸引子，使它可以用做联想记忆存储器，即合理选择权系数，使网络的稳态为一组状态 X。稳态应该为 X 与初态在 Hamming 距离上最近的状态。

本例是研究 BAM 网络在飞机空中特情处理中的应用。由于 MATLAB 没有为 BAM 网络提供专门的函数，所以只有通过 MATLAB 给定的算法来解决这个问题。

1. 问题描述

下面以飞机的发动机为例，设发动机喘振的标志量为故障代码 1（C1），发动机冒烟的故障代码 2（C2），有爆音的标志量为故障代码 3（C3）。

设飞机故障向量为发动机停车（E1），压缩器喘振（E2），发动机失火（E3），双转子变为单转子（E4）。当上述故障成立时，它的值为二值量 1。

设故障特征参数向量为发动机高压转速 N2，发动机低压转速 N1，排气温度 T4，飞行速度 V，火警灯亮 Hj，故障代码 1（C1），故障代码 2（C2），故障代码 3（C3）。其中，Hj、C1、C2、C3 为二值量，1 表示成立，0 表示不成立。N2、N1、T4、V 连续下降为 1，连续上升为 0（编码时这些参数为二值量）。

按照以下的方式定义处置措施向量[M1,M2,M3,M4]。

（E1,M1）：（1）将油门收到停车位置；（2）检查高度，向就近机场下滑；（3）检查补氧压力，做好开车准备。

（E2,M2）：（1）如果推油门杆时产生喘振，应立即收油门杆到慢车位置，若喘振停止则推油门杆时应柔和；若喘振不停止则应停车，然后按规定进行空中开车。（2）如果由于进气道喘振而引起压缩器喘振，应断开加力，将放气门操纵电门扳到向下，并检查转速和喷气温度的指示情况。（3）如果飞机进入螺旋而引起喘振，应首先收油门杆到慢车位置，集中精力改出螺旋，待改出螺旋后再检查发动机的工作情况。如果发动机已停车，应收油门杆到停车位置 3~4s 以上，以吹除燃烧室内的积油，然后再进行空中开车。

（E3,M3）：（1）收油门杆到停车位置；（2）关闭一、二、三油泵；（3）用上升的方法减速到 400~500Km/h；（4）按灭火按钮，灭火后，禁止空中开车；（5）如没有条件迫降或灭火不成功应跳伞。

（E4,M4）：（1）保持转速不小于 85%，移动油门杆应特别柔和，以免造成发动机喘振停车；（2）立即返场，四转弯后仍保持转速不小于 85%，可用减速板调整下滑速度，判明确有把握进入跑道时，用收油门杆关车的方法修正目测；（3）如已造成发动机熄火停车时，则应按当时实际情况决定跳伞或迫降。

编码方案：故障向量 y=[E1 E2 E3 E4]，故障特征参数向量为 x=[N2 N1 T4 Hj C1 C2 C3]，处置措施向量为 z=[M1 M2 M3 M4]。则有：

y_1=[1 0 0 0],x_1=[1 1 1 1 0 0 0 1],z_1=[1 0 0 0]
y_2=[0 1 0 0],x_2=[1 1 0 0 1 0 0 0],z_2=[0 1 0 0]
y_3=[0 0 1 0],x_3=[0 0 0 0 1 1 1 0],z_3=[1 0 0 0]

2. 仿真运算

实现联想的关键是充分训练联想权阵，使网络达到稳定。在确保网络稳定的前提下，首先形成联想矩阵，由 Hebb 学习规则将所有的向量转化为双极性的形式，有：

y_1=[1 -1 -1 -1],x_1=[1 1 1 1 -1 -1 -1 1],z_1=[1 -1 -1 -1]
y_2=[-1 1 -1 -1],x_2=[1 1 -1 -1 1 -1 -1 -1],z_2=[-1 1 -1 -1]
y_3=[-1 -1 1 -1],x_3=[-1 -1 -1 -1 1 1 1 -1],z_3=[1 -1 -1 -1]

根据联想矩阵的定义，可以利用 MATLAB 数学计算功能得到联想矩阵 W。实验证明，联想矩阵经过充分的学习和训练后，可以达到较好的效果。

$$W^T=\left[\sum_{k=1}^{3}x_k^T y_k\right]^T=\begin{bmatrix}1 & 1 & 3 & 3 & -3 & -1 & -1 & 3\\ 1 & 1 & -1 & -1 & 1 & -1 & -1 & -1\\ -3 & -3 & -1 & -1 & 1 & 3 & 3 & -1\\ -1 & -1 & 1 & 1 & -1 & 1 & 1 & 1\end{bmatrix}$$



给定训练样本后，就可以对网络进行训练。

4.6 回归 BP 网络及应用

误差反向传播 BP 算法是前向网络学习算法中应用最为广泛的算法，因此，一些研究者尝试将 BP 算法中采用的梯度下降法推广到回归网络中，由此产生了回归 BP（Recurrent BP）网络。

4.6.1 回归 BP 网络概述

回归 BP 网络同时具有反馈和前馈机制，这就意味着在网络的一个训练周期中，网络的输出同时反馈给网络的输入神经元作为网络的外部输入。如图 4-24 所示，为一个典型的三层回归 BP 网络。

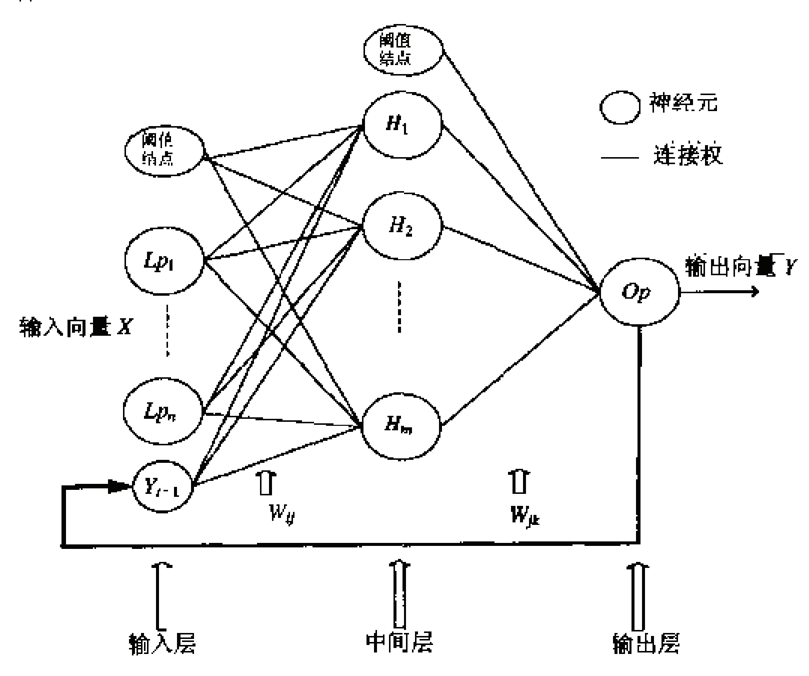


图 4-24 回归 BP 网络

在图 4-24 中，输入层有一个 n 维的输入向量和一个阈值结点，该结点的值是固定的，这个值的存在保证了网络的收敛特性。中间层有 m 个神经元和一个阈值结点。相邻两层的所有神经元采用全连接的方式相连。输入层的神经元输入输出关系可以表示为：

$$I_i^F = O_i^I = X_i \qquad i = 1, 2, \cdots, n$$

$$O_i^F = f\left(I_i^F\right) = f\left(X_i\right)$$

其中，F 表示输入层，输入神经元和隐含神经元之间实现加权连接。即如果信号从第 i 个神经元传递到第 j 个神经元，则信号需要乘上两个神经元之间的连接权值 w_{ij}。令 O_i 表示第 i 个神经元的输出，第 j 个神经元的输入则为 O_iW_{ij}。对第 j 个神经元的输入进行求和，可得



$$I_j^H = \sum_{i=1}^{n_1} O_i^F W_{ij}^F + \theta_j^H \qquad j = 1,2,\cdots,m$$

式中，θ_j 为阈值项，H 表示中间层。这种加法操作是通过中间层的处理器实现的，实现加法的过程就是激发神经元的过程。由于神经元的权值和输入可以取正值，也可以取负值，因此对神经元的激发有可能产生正值、负值和零值中的任意数据。中间层的激发函数为：

$$O_j^H = f\left(I_j^H\right) = f\left(\sum_{i=1}^{n_1} O_i^F W_{ij}^F + \theta_j^H\right) \qquad j = 1,2,\cdots,m$$

对于输出层 Y，它的第 k 个神经元接收了中间层第 j 个神经元的输出信号，经过加权后，作为自己的输入信号，可以得到同上面类似的结论。

$$I_k^Y = \sum_{j=1}^{n_1} O_j^H W_{jk}^H + \theta_k^Y \qquad k = 1,2,\cdots,g$$

$$O_k^Y = f\left(I_k^Y\right) = f\left(\sum_{j=1}^{n_1} O_j^H W_{jk}^H + \theta_k^Y\right) \qquad k = 1,2,\cdots,g$$

回归 BP 网络的传递函数见下式。这是一种非常典型的函数，具有很多优点。它是连续可导的，输出也是连续的并且位于区间[0,1]中。如图 4-25 所示，β 为函数的斜率。

$$O_k^Y = f\left(net\right) = \frac{1}{1+\exp\left(-\beta * net\right)} \qquad \beta > 0$$

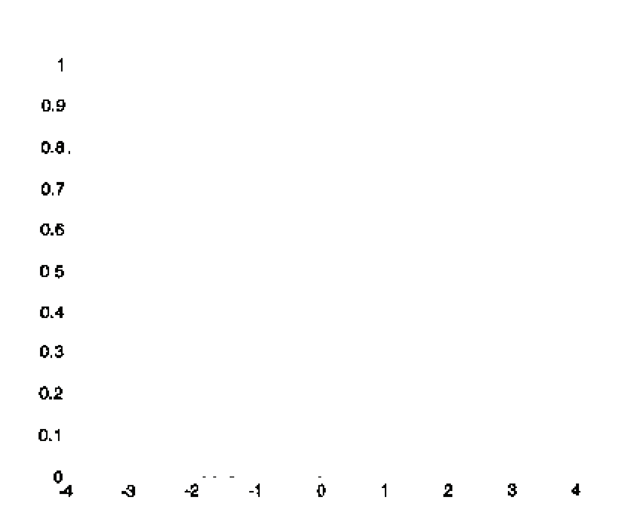


图 4-25　回归 BP 网络传递函数曲线（β=1）

4.6.2　基于回归 BP 网络的房价预测

房价及其变化趋势是很多人关注的焦点，也是制定合理的购房策略的基础。现以某城



市的房价预测为例，展开基于回归 BP 网络的预测研究。

一般来说，房价受宏观调控政策、房产和人口等多方面的影响，可以将房价写为如下的函数模型：

$$RHP = f\left(Q_d, Q_s, t\right) = f\left(x_i, y_i, z_i, v_i, t\right)$$

其中，RHP 表示房价，t=1,2,…,n，i=1,2,…,m。Q_d表示在 t 时间段中对房子的总体需求，Q_s表示在 t 时间段中房子的总体供应。x 表示宏观经济变量，如 GDP 等。y 表示与买方相关的变量，如家庭平均收入等。z 表示与人口统计相关的变量，如人口比例、新婚人口数目等。v 表示与卖方相关的变量，如建房费用和地皮价格等。x、y、x 和 z 是完全无关的变量。

由于房价受多方面因素的影响，在建立一个房价预测模型时，需要综合考虑多种因素。而 BP 算法在处理多输入的非线性系统的过程中表现出了相当强的能力，所以这里采用 BP 算法。之所以采用递归 BP 网络，是因为房价是一种时间序列信号，目前的数据可能对以后的数据产生重要的影响。

由于 MATLAB 神经网络工具箱中没有为递归 BP 网络提供专门的工具，所以这里只给出基于递归 BP 网络进行房价预测的有关思想，具体算法读者可参见有关的资料，并利用 MATLAB 的数学计算功能实现。

4.7 Boltzmann 机网络及仿真

Hinton 等人在 1985 年将模拟退火算法引入到神经网络中，提出了 Boltzmann 机网络，简称 BM 网络。

4.7.1 BM 网络的基本结构

BM 网络结构与离散 Hopfield 网络结构相似，由 N 个神经元组成，每个神经元取 0-1 二值输出，且神经元之间以对称连接权相互连接。与 Hopfield 网络不同的是，BM 网络通常将整个神经元分为可视层和隐含层两大部分。可视层又可分为输入部分和输出部分，但它与一般阶层网络的区别在于它没有明显的层次界限，且神经元之间不是单向连接而是双向连接的，如图 4-26 所示。

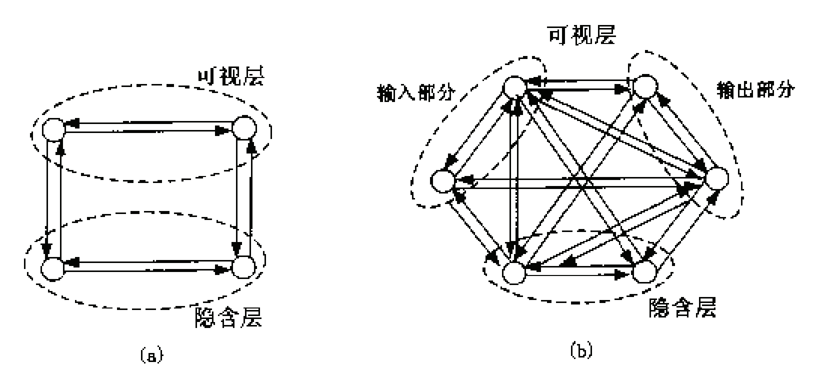


图 4-26　BM 网络结构



4.7.2 BM 模型的工作规则和学习规则

BM 网络的算法根据其用途可以分为工作规则和学习规则。工作规则也就是网络的状态更新规则，主要用于优化组合问题。学习规则就是连接权值和输出阈值的修正规则，主要用于将网络作为一个外界概率分布的模拟机的场合。这也是 BM 网络的一个独特的方面。首先介绍 BM 网络的工作规则。

实际上，BM 网络的工作规则就是模拟退火算法的具体体现，其步骤如下。

假定网络有 N 个神经元，各神经元之间的连接权值为 w_{ij}，各神经元的输出阈值为 θ_i，输出为 u_i，神经元 i 的内部状态为 H_i，$i,j=1,2,\cdots,N$。网络温度 $T|_{t=0}=T_0$，为 w_{ij} 和 θ_i 赋予区间[－1,1]之间的随机值，并令 $w_{ij}=w_{ji}$。

（1）从 N 个神经元中随机选取一个神经元 i。

（2）按照下式求出神经元 i 的输入总和，即内部状态 H_i：

$$H_i(t)=\sum_{j=1,j\neq t}^{N} w_{ij}u_j(t)-\theta_i$$

（3）按照下式得到的概率将神经元状态更新为 1：

$$P\left[u_i(t+1)=1\right]=\frac{1}{1+\exp\left(-H_i(t)/T\right)}$$

（4）i 以外的神经元的输出状态保持不变：

$$u_j(t+1)=u_j(t) \qquad j=1,2,\cdots,N, \quad j\neq i$$

（5）令 $t=t+1$，按照下式计算新的温度参数 $T(t+1)$：

$$T(t+1)=\frac{T_0}{\log(t+1)}$$

（6）返回到步骤（1），直到温度参数 T 小于预先设定的截止温度 T_d。

在步骤（3）中更新网络状态，一般有两种方法。

（1）$H_i(t)>0$ 时，直接令 $u_i(t+1)=1$；$H_i(t)<0$ 时，产生一个位于区间[0,0.5]内的随机数 $\varepsilon(t)$；当 $P\left[u_i(t+1)=1\right]>\varepsilon$ 时，令 $u_i(t+1)=1$，否则令 $u_i(t+1)=u_i(t)$。

（2）$H_i(t)>0$ 时，直接令 $u_i(t+1)=1$；$H_i(t)<0$ 时，当 $P\left[u_i(t+1)=1\right]$ 大于预先设定的概

率值ε（$\varepsilon<0.5$）时，令$u_i(t+1)=1$，否则令$u_i(t+1)=u_i(t)$。

此外，关于初始温度T_0和结束温度T_E，一般凭经验给出。

BM 网络除了可以解决优化组合问题外，还可以通过网络训练模拟外界给出的概率分布，实现概率意义下的联想记忆。联想记忆分为自联想记忆和互联想记忆两种模式。当把一组记忆模式及其概率分布函数提供给 BM 网络的可视层后，网络按照一定的学习规则进行学习，学习结束后，当网络状态按照工作规则进行不断转移时，网络的各个状态间按照记忆的学习模式的概率分布出现，即概率大的状态出现的频率高，概率小的状态出现的频率低。这种概率意义下的联想记忆就称为自联想记忆。

如果将某个记忆模式提供给网络的输入部分，同时，在输出部分按照给定的概率分布给出一组目标输出模式。此时给出的概率分布函数实际上是输出模式相对于输入模式的条件概率分布。BM 网络正是通过记忆这种条件概率分布函数来完成互联想记忆的。例如，由 BM 网络对柴油机进行故障诊断，当为网络提供一个"排气筒有黑烟"的故障模式后，在网络的输出部分按产生这种故障现象的原因的概率大小提供一系列的输出模式，如汽缸点火位置不对、油料中有杂质等。这样就构成了网络的学习模式样本。

无论是自联想记忆还是互联想记忆，其实质都是通过学习目标概率分布函数，将其记忆并在以后的回想过程中将这一概率分布再现出来。

下面首先介绍自联想记忆学习规则。假设网络的共有N个神经元，其中可视层有n个神经元，隐含层有$m=N-n$个神经元。可视层有$p=2^n$个状态，隐含层有$q=2^m$个状态，整个网络则有$M=2^N$个状态。各层的状态可表示为：可视层状态$U_a=(u_1,u_2,\cdots,u_n)$，隐含层状态$U_b=(u_1,u_2,\cdots,u_m)$，其中$a=1,2,\cdots,p$，$b=1,2,\cdots,q$。整个网络的状态的概率分布函数为$Q(U_a,U_b)$，网络的连接权值和输出阈值分别为w_{ij}和θ_j，i、$j=1,2,\cdots,N$，网络在第k个状态时的能量为$E_k(U_a,U_b)$。学习过程如下。

（1）初始化。将连接权w_{ij}赋予区间[－1,1]之间的随机值并令$\theta_j=0$。

（2）按给定的外界概率（目标概率分布）$P(U_a)$将网络可视层的各神经元固定在某一状态$U_a=(u_1,u_2,\cdots,u_n)$。

（3）从温度T_0开始，按照网络工作规则（即模拟退火算法）对网络隐含层的各神经元的输出进行状态更新，直到达到温度T_d下的平衡状态$U_b=(u_1,u_2,\cdots,u_m)$。

（4）在隐含层的平衡状态下，保持温度T_d不变，再进行L次全网络的状态更新，每次更新后，当神经元i和j同时为1时，计算下式（学习过程）：



$$n_{ij}^{+}=n_{ij}^{+}+1$$

（5）重新从温度 T_0 开始，按照网络工作规则对全网络神经元状态 u_j 进行更新，直到达到温度 T_d 下的平衡状态 $U=(u_1,u_2,\cdots,u_n,u_{n+1},u_{n+2},\cdots,u_{n+m})$， $n+m\leqslant N$。

（6）在网络的平衡状态下，保持温度 T_d 不变，再进行 L 次全网络的状态更新，每次更新后，当神经元 i 和 j 同时为 1 时，计算下式（反学习过程）：

$$n_{ij}^{-}=n_{ij}^{-}+1$$

（7）返回步骤（2），一共进行 M 次循环，并要求 $M>p$，p 为可视层的状态个数。

（8）计算对称概率 P_{ij}^{+},P_{ij}^{-}：

$$P_{ij}^{+}=\frac{1}{M\times L}n_{ij}^{+},\quad P_{ij}^{-}=\frac{1}{M\times L}n_{ij}^{-}\qquad i、j=1,2,\cdots,N$$

（9）按照下式调整网络的连接权值 w_{ij}：

$$w_{ij}=w_{ij}+\varepsilon\left(P_{ij}^{+}-P_{ij}^{-}\right)/T_d\qquad i、j=1,2,\cdots,N$$

（10）返回步骤（2），直到循环次数大于或等于预先设定的值。

下面介绍互联想记忆学习准则。在以前假设条件的基础上，进一步假设可视层输入部分有 n_i 个神经元，对应 $p_i=2^{n_i}$ 个状态 $U_a=(u_1,u_2,\cdots,u_{n_i})$；输出部分有 n_0 个神经元，对应 $p_0=2^{n_0}$ 个状态 $L_c=(l_1,l_2,\cdots,l_{n_0})$。

目标联合概率分布 $P(U_a,L_c)=P(U_a)P(L_c|U_a)$。学习步骤如下。

（1）初始化。将连接权 w_{ij} 赋予区间[－1,1]之间的随机值并令 $\theta_j=0$。

（2）随机选取输入模式 $U_a=(u_1,u_2,\cdots,u_{n_i})$ 提供给可视层的输入部分。

（3）按照目标条件概率分布 $P(L_c|U_a)$，将输出模式 $L_c=(l_1,l_2,\cdots,l_{n_0})$ 固定在可视层的输出部分上。

（4）从温度 T_0 开始，按照网络工作规则（即模拟退火算法）对网络隐含层的各神经元的输出进行状态更新，直到达到温度 T_d 下的平衡状态。

（5）在隐含层的平衡状态下，保持温度 T_d 不变，再进行 L 次全网络的状态更新，每次更新后，当神经元 i 和 j 同时为 1 时，计算下式（学习过程）：

$$n_{ij}^{+} = n_{ij}^{+} + 1$$

（6）重新从温度 T_0 开始，按照网络工作规则对全网络神经元状态 u_i 进行更新，直到达到温度 T_d 下的平衡状态。

（7）在网络的平衡状态下，保持温度 T_d 不变，再进行 L 次全网络的状态更新，每次更新后，当神经元 i 和 j 同时为 1 时，计算下式（反学习过程）：

$$n_{ij}^{-} = n_{ij}^{-} + 1$$

（8）返回步骤（3），一共进行 M_1 次循环，并要求 $M_1 > p_0$，p_0 为可视层输出部分的的状态个数。

（9）计算对称概率 P_{ij}^{+}, P_{ij}^{-}：

$$P_{ij}^{+} = \frac{1}{M \times L} n_{ij}^{+}, \quad P_{ij}^{-} = \frac{1}{M \times L} n_{ij}^{-} \qquad i、j = 1, 2, \cdots, N$$

（10）按照下式调整网络的连接权值 w_{ij}：

$$w_{ij} = w_{ij} + \varepsilon \left(P_{ij}^{+} - P_{ij}^{-} \right) \Big/ T_d \qquad i、j = 1, 2, \cdots, N$$

（11）返回步骤（2），选取下一组学习模式进行 M_2 次循环，且要求 $M_2 > p_1$，p_1 为可视层输入部分的状态个数。

（12）从步骤（2）～（11）进行 Y 次循环后学习结束，其中 Y 为预先设定的最大循环次数。



在对每一组学习模式进行训练时，输入部分的状态总是固定在某个输入模式的状态。

在学习结束后网络进行回想时，当给网络的输入部分提供一输入模式 U_a 后，对网络输入部分按照网络工作规则进行状态更新，在网络的输出部分各个状态出现的概率将符合学习过的希望概率分布 $P(L_c|U_a)$。

4.7.3 BM 网络的 MATLAB 仿真

这里利用一个非常简单的例子来说明 BM 网络的自联想学习规则。设网络只有两个神经元，一个为可视层神经元 V_0，一个为隐含层神经元 V_1。两个神经元各有 0-1 两种状态，可视层输出为 1 的概率为 0.1。网络结构如图 4-27 所示。

图 4-27 含有两个神经元的 BM 网络结构

MATLAB 神经网络工具箱中没有关于 BM 网络的函数，因此，需要利用 MATLAB 的数学计算功能来实现 BM 网络的学习和训练。

首先赋予网络权值 $w_{10}=-10$，$w_{01}=-10$。设网络温度为 $T_0=100$，$T_d=10$，以快速降温的方式，按照网络工作规则对网络进行状态更新，以得到进行学习和训练之前的状态概率分布，即网络状态的初始分布情况，如表 4-3 所示。

表 4-3　网络状态的初始分布

网络状态		对应能量	状态出现的理论概率	状态出现的实际概率
可视层	隐含层			
0	0	0.0000	0.0000	0.1310
0	1	-8.0000	0.0000	0.2530
1	0	-10.0000	0.0000	0.3580
1	1	-8.0000	0.0000	0.2580

网络状态对应的能量按照下式计算：

$$E_i=-\frac{1}{2}\sum_{\substack{j=1\\j\neq i}}^{N}w_{ij}u_iu_j+\theta_i u_i$$

其中，N 为网络状态的个数，这里 $N=4$。

网络各个状态出现的理论概率按照下式计算：

$$Q(E_i)=\frac{\exp(-E_i/T)}{Z},\quad Z=\sum_{i=1}^{N}\exp(-E_i/T)$$

由表 4-3 可得网络可视层输出为 1 的概率为 0.3580+0.2580=0.6160。

接下来进行网络学习，已知 $P(U_a)=P(U_0=1)=1$。设 $T_0=100$，$T_d=10$，$L=1000$，$M=128$，$Y=6$，$\varepsilon=0.1$。如表 4-4 所示为每次大循环后，网络可视层输出为 1 的概率分布。

表 4-4　网络可视层输出为 1 的概率分布

1	2	3	4	5	6
0.3190	0.3190	0.2260	0.2260	0.1590	0.1390

网络学习结束后，以网络温度 $T_0=100$，$T_d=10$ 的快速降温方式，按照网络工作规则，令网络进行回想。网络连接权的初始值为 $w_{10}=-16.7656$，$w_{01}=-16.7656$，网络的回想结果如表 4-5 所示。

表 4-5　网络回想结果

网络状态		对应能量	状态出现的理论概率	状态出现的实际概率
可视层	隐含层			
0	0	0.0000	0.4922	0.1460
0	1	-16.9858	0.4219	0.7560
1	0	7.8008	0.0156	0.0430
1	1	7.5806	0.0703	0.0550

由表 4-5 可得网络可视层输出为 1 的概率为 0.0430+0.0550=0.0980。可见网络的实际概率分布和希望概率分布 0.1 已经十分接近。

4.8 小　　结

本章介绍了反馈型神经网络及其 MATLAB 实现。反馈型神经网络主要包括 Elman 网络、Hopfield 网络、CG 网络、BSB 网络、BAM 网络、回归 BP 网络和 Boltzmann 机网络等。反馈型网络的应用领域主要是优化和预测，其中，Elman 网络和回归 BP 网络主要应用于预测，而 Hopfield 网络等则主要应用于优化。

神经网络工具箱中只提供了有关 Elman 网络和 Hopfield 网络的函数，因此，利用这两种网络来解决实际问题是比较方便的。本章为 Elman 网络提供了一个空调负荷预测的实例，为 Hopfield 网络提供了一个字符识别的实例，读者可参照这两个实例来学习利用神经网络解决实际问题的过程。

第5章 自组织与LVQ神经网络理论及MATLAB实现

自组织神经网络是一类无教师学习的神经网络模型，这类模型大都采用竞争型的学习规则。自组织神经网络无需提供教师信号，它可以对外界未知环境（或样本空间）进行学习或者仿真，并对自身的网络结构进行适当调整，这就是所谓自组织。

本章将介绍以下内容：

- 自组织竞争网络及 MATLAB 实现
- 自组织特征映射神经网络及 MATLAB 实现
- 自适应共振理论模型及 MATLAB 实现
- LVQ 神经网络及 MATLAB 实现
- CPN 模型及 MATLAB 实现

5.1 自组织竞争网络及 MATLAB 实现

竞争型神经网络的基本思想是网络竞争层的各神经元通过竞争来获取对输入模式的响应机会，最后仅有一个神经元成为竞争胜利者，并将与获胜神经元有关的各连接权值向着更有利于其竞争的方向调整。自组织竞争网络自组织、自适应的学习能力进一步拓宽了神经网络在模式分类和识别方面的应用。

5.1.1 基本竞争型神经网络概述

竞争型神经网络有很多具体形式和不同的学习算法，本节只介绍一种比较简单的网络结构和学习算法。网络结构如图 5-1 所示。

竞争型网络可分为输入层和竞争层。假定输入层由 N 个神经元构成，竞争层有 M 个神经元。网络的连接权值为 w_{ij}， $i=1,2,\cdots,N$， $j=1,2,\cdots,M$ ，且满足约束条件 $\sum_{i=1}^{N} w_{ij}=1$ 。在竞争层中，神经元之间相互竞争，最终只有一个或者几个神经元获胜，以适应当前的输入样本。竞争胜利的神经元就代表着当前输入样本的分类模式。

竞争型网络的输入样本为二值向量，各元素取值 0 或 1。竞争层神经元 j 的状态可按下式计算：

$$s_j = \sum_{i=1}^{N} w_{ij} x_i$$

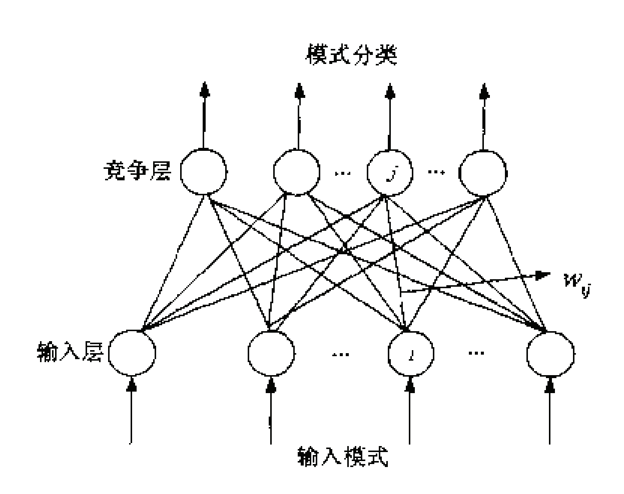


图 5-1　基本竞争型神经网络结构

其中，x_i 为输入样本向量的第 i 个元素。根据竞争机制，竞争层中具有最大加权值的神经元 k 赢得竞争胜利，输出为：

$$a_k = \begin{cases} 1 & s_k > s_j, \quad \forall j, \quad k \neq j \\ 0 & 其他 \end{cases}$$

竞争后的权值按照下式进行修正，对于所有的 i，有：

$$w_{ij} = w_{ij} + \alpha \left(\frac{x_i}{m} - w_{ij} \right)$$

其中，α 为学习参数，$0 < \alpha << 1$，一般取为 0.01~0.03；m 为输入层中输出为 1 的神经元个数，即 $m = \sum_{i=1}^{N} x_i$。

权值调整公式中的 $\frac{x_i}{m}$ 项表示当 x_i 为 1 时，权值增加；而当 x_i 为 0 时，权值减小。也就是说，当 x_i 活跃时，对应的第 i 个权值就增加，否则就减小。由于所有权值的和为 1，所以当第 i 个权值增加或减小时，对应的其他权值就可能减小或增加。此外，该公式还保证了权值的调整能够满足所有的权值调整量之和为 0。

5.1.2　自组织竞争网络的应用

本例采用自组织竞争网络，完成测井资料的岩性识别。神经网络通过对已知井段测井

数据进行学习，来预测同一地区其他井段的岩性。

1. 问题描述

岩性识别是储层评价的重要工作之一，是求解储层参数的基础。利用测井资料进行岩性识别有很多方法，神经网络方法是一种应用比较广泛的方法之一。

本例选择某地区的资料进行研究。该地区属于碳酸盐地层，因此需要判断的岩性有 3 种，即泥岩、砂岩和石灰岩。通过对历史资料的分析和现场试验可知，影响岩性有 5 个重要的因子，即补偿中子空隙度 CNL、补偿密度曲线 DEN、声波时差 DTC、自然伽玛 GR 和微电阻率 RT。

通过对历史资料进行分析，获得了 100 组样本点，由于篇幅的原因，只列出其中的 6 组，如表 5-1 所示。接下来就利用这 6 组样本作为网络的训练样本。

表 5-1　岩性影响因子（数据已经归一化）

序　号	CNL	DEN	DTC	GR	RT	岩　性
1	0.4036	0.4365	0.4860	0.5161	0.3419	泥岩
2	0.4154	0.4711	0.4639	0.4981	0.3806	
3	0.5352	0.6408	0.6145	0.6299	0.7154	砂岩
4	0.5524	0.6528	0.6234	0.6715	0.7025	
5	0.7709	0.7812	0.8204	0.8425	0.8622	石灰岩
6	0.7589	0.7965	0.8125	0.8506	0.8709	

2. 网络创建和训练

利用函数 newc 创建一个自组织竞争网络。由于需要区分的类别数目为 3，因此，神经元的数目也为 3。为了加快学习速度，将学习速率设置为 0.1。MATLAB 代码为：

```
P=[ 0.4036    0.4154    0.5352    0.5524    0.7709    0.7589;
    0.4365    0.4711    0.6408    0.6528    0.7812    0.7965;
    0.4860    0.4639    0.6145    0.6234    0.8204    0.8125;
    0.5161    0.4981    0.6299    0.6715    0.8425    0.8506;
    0.3419    0.3806    0.7154    0.7025    0.8622    0.8709];
net=newc(minmax(P),3,0.1);
```

网络创建结束后，接下来需要对网络进行训练，MATLAB 代码为：

```
net=init(net);
net.trainParam.epochs=200;
net=train(net,P');
```

训练结果为：

```
TRAINR, Epoch 0/200
TRAINR, Epoch 25/200
TRAINR, Epoch 50/200
TRAINR, Epoch 75/200
TRAINR, Epoch 100/200
TRAINR, Epoch 125/200
TRAINR, Epoch 150/200
TRAINR, Epoch 175/200
```

```
TRAINR, Epoch 200/200
TRAINR, Maximum epoch reached.
```

由此可见，当达到最大训练次数时，训练停止。此时为了检验网络的分类性能，需要对网络进行测试。利用仿真函数检验网络对上述岩性模式的分类。

```
Y=sim(net,P');
```

运行结果为：

```
Y =
    (2,1)        1
    (2,2)        1
    (1,3)        1
    (1,4)        1
    (3,5)        1
    (3,6)        1
```

利用函数 vec2ind 将 Y 转换为串行数据：

```
Yc=vec2ind(Y);
```

结果为：

```
2    2    1    1    3    3
```

由此可见，网络成功地对上述岩性模式进行了分类，其中 P 的前两组数据（列向量）为一类，中间两组数据为一类，最后两组数据为一类，这与表 5-1 中的数据是吻合的。

我们知道，采用训练样本以外的数据对网络进行测试，是一种最好的测试方案。现有一组石灰岩的影响因子，接下来利用该组数据对网络进行测试，看网络能否成功地对它进行识别。

```
P_test=[0.7601 0.8123 0.8079    0.8450    0.8792]';
Y_test=sim(net,P_test);
Yc_test=vec2ind(Y_test)
```

结果为：

```
Yc_test=3
```

结果说明该组数据属于第 3 类，即石灰岩。由此可见，网络成功地识别了该组数据，因此可以说，网络的性能是不错的。

5.2 自组织特征映射（SOM）神经网络及 MATLAB 实现

自组织特征映射网络也称为 Kohonen 网络，或者称为 Self-Organizing Feature Map（SOM）网络，它是由芬兰学者 Teuvo Kohonen 于 1981 年提出的。该网络是一个由全连接的神经元阵列组成的无教师自组织、自学习网络。Kohonen 认为，处于空间中不同区域的神经元有不同的分工，当一个神经网络接受外界输入模式时，将会分为不同的反应区域，各区域对输入模式具有不同的响应特性。



5.2.1 SOM 网络的结构

SOM 网络结构如图 5-2 所示。

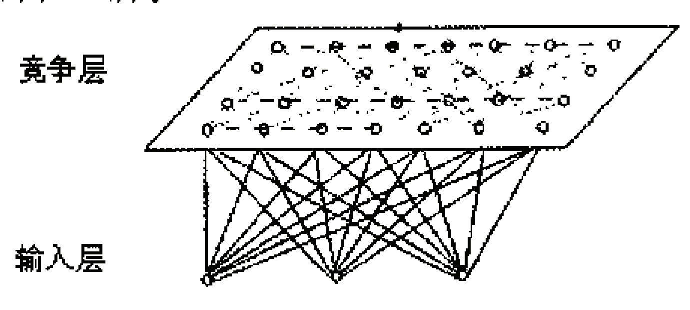


图 5-2 SOM 网络的结构

SOM 网络的一个典型特性就是可以在一维或二维的处理单元阵列上，形成输入信号的特征拓扑分布，因此 SOM 网络具有抽取输入信号模式特征的能力。SOM 网络一般只包含有一维阵列和二维阵列，但也可以推广到多维处理单元阵列中去。下面只讨论应用较多的二维阵列。SOM 网络模型由以下 4 个部分组成。

（1）处理单元阵列。用于接受事件输入，并且形成对这些信号的“判别函数”。

（2）比较选择机制。用于比较“判别函数”，并选择一个具有最大函数输出值的处理单元。

（3）局部互连作用。用于同时激励被选择的处理单元及其最邻近的处理单元。

（4）自适应过程。用于修正被激励的处理单元的参数，以增加其对应于特定输入“判别函数”的输出值。

假定网络输入为 $X \in R^n$，输出神经元 i 与输入单元的连接权值 $W_i \in R^n$，则输出神经元 i 的输出 o_i 为：

$$o_i = W_i X$$

网络实际具有响应的输出单元 k，该神经元的确定是通过“赢者通吃”的竞争机制得到的，其输出为：

$$o_k = \max_i \{o_i\}$$

以上两式可修正为：

$$o_i = \sigma\left(\varphi_i + \sum_{t \in S_i} r_k o_t - o_k\right), \quad \varphi_i = \sum_{j=1}^{m} w_{ij} x_j, \quad o_k = \max_i \{o_i\} - \varepsilon$$

其中，w_{ij} 为输出神经元 i 和输入神经元 j 之间的连接权值。x_i 为输入神经元 i 的输出。$\sigma(t)$ 为非线性函数，即

$$\sigma(t)=\begin{cases}0 & t<0\\ \sigma(t) & 0\leqslant t\leqslant A\\ A & t>A\end{cases}$$

ε 为一个很小的正数，r_k 为系数，它与权值及横向连接有关。S_i 为与处理单元 i 相关的处理单元集合，o_k 称为浮动阈值函数。

5.2.2 SOM 网络学习算法

SOM 网络的学习算法过程为：

（1）初始化。对 N 个输入神经元到输出神经元的连接权值赋予较小的权值。选取输出神经元 j 个“邻接神经元”的集合 S_j。其中，$S_j(0)$ 表示时刻 $t=0$ 的神经元 j 的“邻接神经元”的集合，$S_j(t)$ 表示时刻 t 的的“邻接神经元”的集合。区域 $S_j(t)$ 随着时间的增长而不断缩小。

（2）提供新的输入模式 X。

（3）计算欧氏距离 d_j，即输入样本与每个输出神经元 j 之间的距离：

$$d_j=\| X-W_j \|=\sqrt{\sum_{i=1}^{N}\left[x_i(t)-w_{ij}(t)\right]^2}$$

并计算出一个具有最小距离的神经元 j^*，即确定出某个单元 k，使得对于任意的 j，都有 $d_k=\min\limits_j\left(d_j\right)$。

（4）给出一个周围的邻域 $S_k(t)$。

（5）按照下式修正输出神经元 j^* 及其“邻接神经元”的权值：

$$w_{ij}(t+1)=w_{ij}(t)+\eta(t)\left[x_i(t)-w_{ij}(t)\right]$$

其中，η 为一个增益项，并随时间变化逐渐下降到零，一般取

$$\eta(t)=\frac{1}{t} \text{或} \eta(t)=0.2\left(1-\frac{t}{10000}\right)$$

（6）计算输出o_k：

$$o_k = f\left(\min_j \| X - W_j \|\right)$$

其中，$f(\cdot)$一般为0-1函数或其他非线性函数。

（7）提供新的学习样本来重复上述学习过程。

5.2.3 基于SOM网络的土壤分类

土壤分类是土壤科学的基础，也是土壤科学发展水平的综合指标。不同时期所拟定的土壤分析系统反映了该时期人们对土壤的认识水平和土壤本身的发展阶段。目前，土壤分类研究已经由单纯的性态描述向指标化和数量化方向发展。模糊聚类分析由于为土壤分类提供了科学的方法而得到广泛的应用。但是在模糊聚类分析中，在进行数据标定时（建立模糊相似矩阵），不同人可以采用不同的方法，例如相关系数法、距离法等。因此建立的模糊相似矩阵是不同的，最后的分类结果也存在局部差异。这说明不同的标定方法所利用、提取的土壤样本间的信息不同，就会影响到最后的结果。另外，在最后确定最佳分类结果即最佳阈值时，通常通过数学统计方法，构造统计量来确定最佳阈值，但该阈值是否合适还需要根据实际问题做进一步分析与判断。这些都说明某种分类方法仅仅提供一种工具和思路， 对于具体问题还要结合实际情况才能做出正确的判断。基于这个问题，这里尝试利用SOM网络进行土壤分类，SOM网络是一种具有聚类功能的神经网络，它为土壤分类提供一种新的思路与方法。

1. 网络样本设计

从资料中得到了我国某地区的10个土壤样本，每个样本用7个理化指标表示其性状，原始数据如表5-2所示。确定网络的输入模式为：

$$P_k = \left(P_1^k, P_2^k, \cdots, P_n^k\right) \quad k = 1, 2, \cdots, 10, \quad n = 7$$

即一共有10组土壤样本向量，每个样本中包括7个元素。

表5-2 土壤样本及性状

序号	土壤类型	全氮（%）	全磷（%）	有机质（%）	pH	代换量	耕层厚（cm）	密度（g/cm^3）
1	薄层黏底白浆化黑土	0.270	0.142	6.46	5.5	35.8	21	1.03
2	厚层黏底黑土	0.171	0.115	3.46	6.3	33.0	60	0.78
3	薄层黏底黑土	0.114	0.101	2.43	6.4	26.5	25	1.13
4	厚层黏底黑土	0.173	0.123	3.30	5.8	28.9	65	1.09
5	薄层黏底黑土	0.145	0.131	3.28	6.0	28.5	25	1.03
6	厚层草甸黑土	0.173	0.140	3.45	5.8	33.4	60	0.98
7	中层草甸黑土	0.250	0.177	5.51	7.2	42.5	45	0.93

（续表）

序号	土壤类型	全氮（%）	全磷（%）	有机质（%）	pH	代换量	耕层厚（cm）	密度（g/cm^3）
8	薄层草甸黑土	0.237	0.189	5.37	6.1	32.9	27	1.00
9	薄层沟谷地草甸黑土	0.319	0.227	7.04	5.8	35.9	24	1.03
10	厚层平地草甸土	0.163	0.124	3.73	6.2	30.6	61	1.28

2. 网络设计

首先利用函数 newsom 创建一个 SOM 网络。代码为：

```
net=newsom(minmax(P),[6 4]);
```

其中，P 为输入向量，由表 5-2 得出，minmax(P)指定了输入向量元素的最大值和最小值，[6 4]表示创建网络的竞争层为 6×4 的结构，网络结构是可以调整的，此处的样本量不是很大，所以选择这样的竞争层是合适的。

然后利用函数 train 和仿真函数 sim 对网络进行训练并仿真。由于训练步数的大小影响着网络的聚类性能。这里设置训练步数为 10、100 和 1000，分别观察其分类性能。

```
a=[10 100 1000 ];
yc=rands(1,10);
for i=1:3
    net.trainParam.epochs=a(i);
    net=train(net,P);
    y=sim(net,P);
    yc=vec2ind(y)
end
```

聚类结果如表 5-3 所示。对结果进行分析可得，当训练步数为 10 时，样本序号为 1、3、5、8 和 9 的分为一类，与表 5-2 进行对比，可知这 5 组样本都是属于薄层的黑土；样本序号为 2、4、6 和 10 的分为一类，而这些都属于厚层的黑土；而序号为 7 的数据单独成为一类，它是中层的黑土。由此可见，网络已经对样本进行了初步的分类，这种分类虽然准确但不够精确。

当训练步数为 100 时，序号为 1 和 9 的样本分为一类，序号为 2、4 和 6 的样本分为一类，序号为 3 和 5 的样本分为一类，序号为 7 的样本为一类，序号为 8 的样本为一类，序号为 10 的样本分为一类，这种分类结果更加细化了。

当训练步数为 1000 时，每一个样本都被划分为一类，这和实际情况也是吻合的。此时如果再提高训练步数，已经没有实际意义了。

表 5-3 聚类结果

训练步数	聚类结果									
10	1	24	1	24	1	24	3	1	1	24
100	19	22	1	12	1	22	4	8	19	24
1000	13	24	1	6	2	23	16	8	20	11

本例的 MATLAB 代码为：

```
P=[
  0.270    0.1426.46  5.5    35.8  21     1.03;
  0.171    0.1153.46  6.3    33.0  60     0.78;
  0.114    0.1012.43  6.4    26.5  25     1.13;
  0.173    0.1233.30  5.8    28.9  65     1.09;
  0.145    0.1313.28  6.0    28.5  25     1.03;
  0.173    0.1403.45  5.8    33.4  60     0.98;
  0.250    0.1775.51  7.2    42.5  45     0.93;
  0.237    0.1895.37  6.1    32.9  27     1.00;
  0.319    0.2277.04  5.8    35.9  24     1.03;
  0.163    0.1243.73  6.2    30.6  61     1.28]';
net=newsom(minmax(P),[6 4]);
a=[10 100 1000];
yc=rands(1,10);
for i=1:3
      net.trainParam.epochs=a(i);
      net=train(net,P);
      y=sim(net,P);
      yc=vec2ind(y)
end
```

5.2.4　基于 SOM 网络的人口分类

人口分类是人口统计中的一个重要指标。由于各方面的原因，我国人口的出生率在性别上的差异比较大，具体表现在同一个时期出生的人口中，一般男的占多数，大大超过了正常的比例。因此，正确地进行人口分类是制定合理的人口政策的基础。

1．样本设计

通过分析历史资料，得到了在 1999 年 12 月共 20 个地区的人口出生比例情况，如表 5-4 所示。

表 5-4　人口出生比例

男（%）	0.5512	0.5123	0.5087	0.5001	0.6012	0.5298	0.5000	0.4965	0.5103	0.5003
女（%）	0.4488	0.4877	0.4913	0.4999	0.3988	0.4702	0.5000	0.5035	0.4897	0.4997

将上表中的数据作为网络的输入样本 P，P 是一个二维随机向量，它的分布情况如图 5-3 所示。

```
P=[0.5512 0.5123    0.5087    0.5001    0.6012    0.5298    0.5000    0.4965    0.5103
  0.5003; 0.4488    0.4877 0.4913    0.4999    0.3988    0.4702    0.5000    0.5035
  0.4897   0.4997];
plot(P(1,:),P(2,:),'+r');
hold on
```

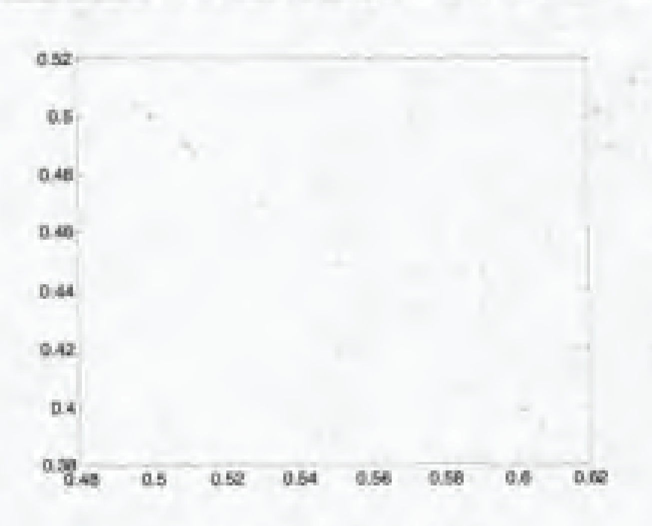


图 5-3　样本数据的分布

2. 网络创建

利用 12 个神经元的 SOM 网络对输入向量 P 进行分类。该网络竞争层神经元的组织结构为 3×4，通过距离函数 linkdist 来计算距离。网络创建代码如下：

```
net=newsom([0 1;0 1],[3 4]);
```

对于该网络，查看它的初始权值：

```
w1_init=net.IW{1,1};
plotsom(w1_init,net.layers{1}.distances);
```

运行结果如图 5-4 所示，图中每一点表示一个神经元，由于网络的初始权值都被设置为 0.5，所以这些点在图中是重合的，看起来就像一个点，实际上是 12 个点。

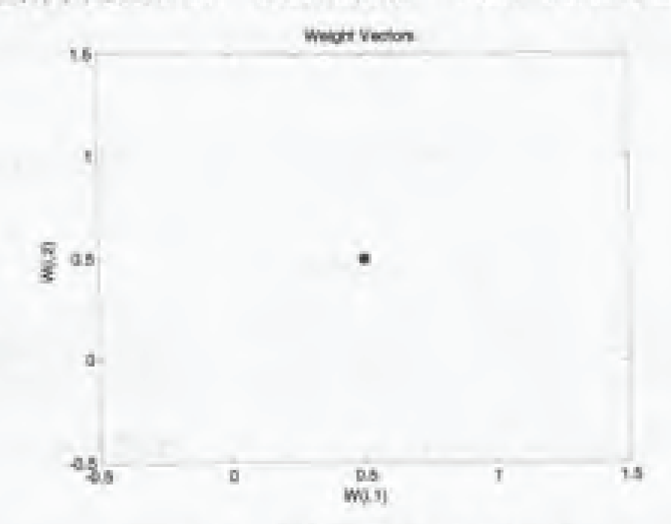


图 5-4　网络初始权值的分布

在命令行窗口中查看 w1_init 的值，可得：

```
w1_init=
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
```

```
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
```

3. 网络训练与测试

接下来利用训练函数 train 对网络进行训练，设想经过训练的网络可对输入向量进行正确分类。网络训练步数对于网络性能的影响比较大，所以这里将步数设置为 100、300 和 500，并分别观察其权值分布。

步数为 100 时的权值分布如图 5-5 所示。

```
%训练步数为 100 时的训练代码
net=train(net,P);
figure;
w1=net.IW{1,1}
plotsom(w1,net.layers{1}.distances);
```

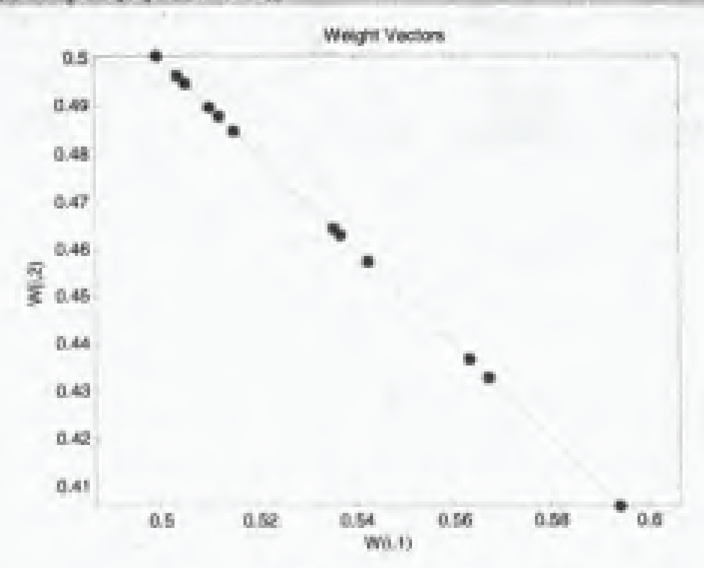


图 5-5　权值分布（训练步数：100）

步数为 300 时的权值分布如图 5-6 所示。

```
%训练步数为 300 时的训练代码
net.trainParam.epochs=300;
net=init(net);
net=train(net,P);
figure;
w1=net.IW{1,1}
plotsom(w1,net.layers{1}.distances);
```

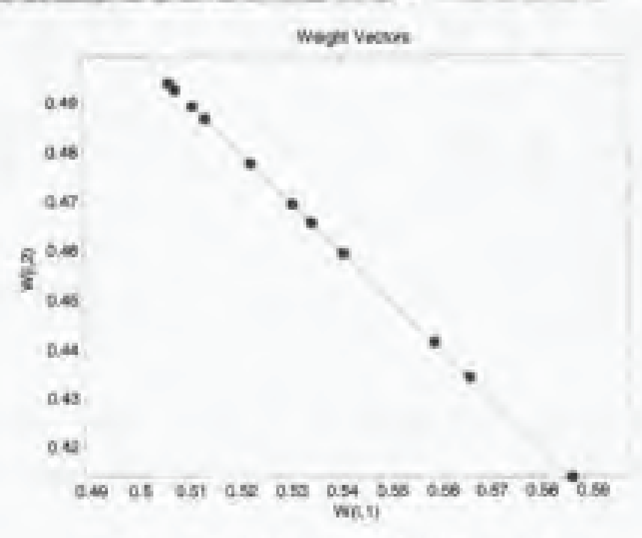


图 5-6　权值分布（训练步数：300）

步数为 500 时的权值分布如图 5-7 所示。

```
%训练步数为 500 时的训练代码
net.trainParam.epochs=500;
net=init(net);
net=train(net,P);
figure;
w1=net.IW{1,1}
plotsom(w1,net.layers{1}.distances);
```

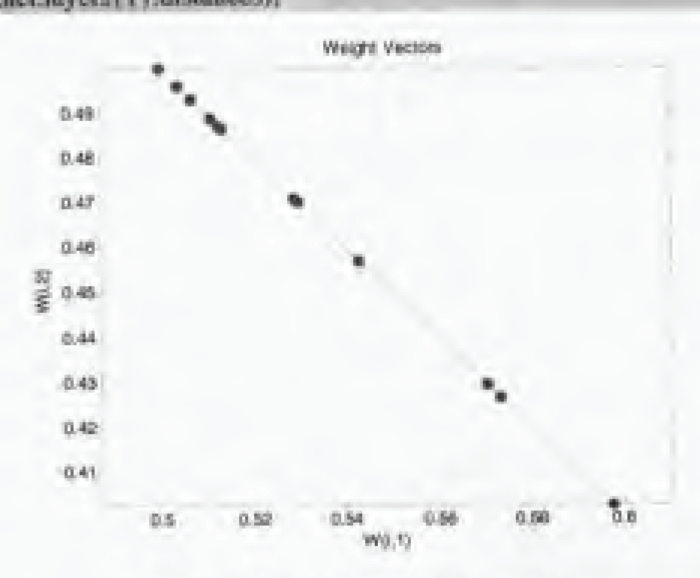


图 5-7　权值分布（训练步数：500）

从图 5-5、5-6 和 5-7 可以看出，训练了 100 步以后，神经元就开始自组织地分布了，每个神经元可以区分不同的样本。随着训练步数的增多，神经元的分布更加合理，但是，当训练次数达到一定值后，权值分布的改变就不很明显了。比如，训练 300 步和训练 500 步后的权值分布就比较相似。

网络训练结束后，权值也就固定了。以后每输入一个值，网络就会自动地对其进行分类。因此，利用这一点对网络进行测试。首先，利用仿真函数 sim 来观察网络对样本数据

的分类结果。

```
Y=sim(net,P);
Y=vec2ind(Y)
```

结果为：

```
Y=
  4    3    10    11    1    7    11    12    6    11
```

对结果进行分析，如表 5-5 所示。

表 5-5 聚类结果

样本序号	类别	激发神经元的索引
1	1	4
2	2	3
3	3	10
4 7 10	4	11
5	5	1
6	6	7
8	7	12
9	8	6

现在，输入一个某地的出生性别比例，检验它属于哪一类。

```
p=[0.5;0.5];
y=sim(net,p);
y=vec2ind(y)
```

结果为 y=11。由此可见，此时激发了网络的第 11 个神经元，所以 p 属于第 4 类。通过直接对比数据可知，p 确实与样本中的第 4 组、第 7 组和第 10 组数据非常接近。

本例的完整 MATLAB 代码为：

```
P=[0.5512 0.5123    0.5087    0.5001    0.6012    0.5298    0.5000    0.4965    0.5103
  0.5003; 0.4488    0.4877 0.4913    0.4999    0.3988    0.4702    0.5000    0.5035
  0.4897   0.4997];
plot(P(1,:),P(2,:),'+r');
hold on
net=newsom([0 1;0 1],[3 4]);
w1_init=net.IW{1,1};
plotsom(w1_init,net.layers{1}.distances);
a=[100 300 500];
for i=1:3
    net=init(net);
    net=train(net,P);
    figure;
    w1=net.IW{1,1};
    plotsom(w1,net.layers{1}.distances);
end
p=[0.5;0.5];
y=sim(net,p);
```



5.3 自适应共振理论模型（ART）及 MATLAB 实现

自适应共振理论英文全称为 Adaptive Resonance Theory，简称 ART，它是由 S.Grossberg 和 A.Carpentent 等人于 1986 年提出的。Grossberg 的研究工作主要是采用数学方法描述人的心理和认知活动，致力于为人类的心理和认知活动建立一个统一的数学模型。以其思想基础提出的 ART 模型成功地解决了神经网络学习中的稳定性（固定某一分类集）和可塑性（调整网络固有参数的学习状态）的关系问题。

ART 是以认知和行为模式为基础的一种无教师、矢量聚类和竞争学习的算法。在数学上，ART 为非线性微分方程的形式；在网络结构上，ART 是全反馈结构，且各层结点具有不同的性质。

ART 网络共有 3 种类型：ART-1、ART-2 和 ART-3。这里主要介绍 ART-1 型网络。

5.3.1 ART-1 型网络模型描述

ART-1 型网络结构如图 5-8 所示。由图可见，网络分为输入和输出两层，一般根据各层所有的功能特征还称输入层为比较层，输出层为识别层。和其他阶层型网络的显著区别是，ATR-1 型网络不仅具有从输入层到输出层的前馈连接权，还有从输出层到输入层的反馈连接权。

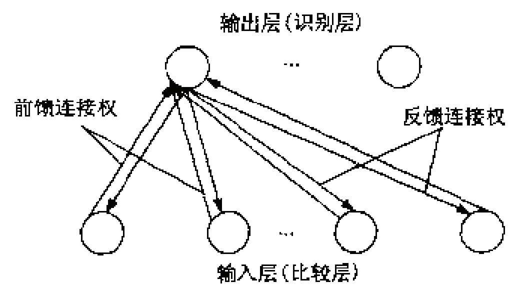


图 5-8 ART-1 型网络结构

假定网络输入层有 N 个神经元，输出层有 M 个神经元，二值输入模式和输出向量分别为 $A_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$ 和 $B_k=\left(b_1^k,b_2^k,\cdots,b_M^k\right)$，其中 $k=1,2,\cdots,p$，p 为输入学习模式的数目。

前馈连接权和反馈连接权分别为 w_{ij} 和 t_{ij}，$j=1,2,\cdots,M$。

ART-1 网络的学习及工作过程，是通过反复地将输入学习模式由输入层向输出层自下而上地识别、由输出层向输入层自上而下地比较来实现的。当这种自下而上的识别和自上而下的比较达到共振，即输入向量可以正确反映输入学习模式的分类，且网络原有记忆没有受到不良影响时，网络对一个输入学习模式的记忆和分类就算完成。网络的学习和工作过程可以分为初始化阶段、识别阶段、比较阶段和探寻阶段，下面将详细介绍网络的学习及工作过程。

5.3.2 ART-1 网络的学习及工作过程

ART-1 网络的学习及工作可以归纳为如下过程。

（1）初始化。令$t_{ij}(0)=1$，$w_{ij}(0)=\dfrac{1}{N+1}$，$i=1,2,\cdots,N$，$j=1,2,\cdots,M$。其中，警戒参数$0<\rho\leqslant 1$。

（2）将输入模式$A_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$提供给网络的输入层。

（3）计算输出层各个神经元的输入加权和：

$$s_j=\sum_{i=1}^{N}w_{ij}a_i^k \quad j=1,2,\cdots,M$$

（4）选择输入模式的最佳分类结果：

$$s_g=\max_{j=1,2,\cdots,M} s_j$$

令神经元 g 的输出为 1。

（5）计算以下 3 式，并进行判断：

$$|A_k|=\sum_{i=1}^{N}a_i^k$$

$$|T_g\cdot A_k|=\sum_{i=1}^{N}t_{gi}a_i^k$$

$$\frac{|T_g\cdot A_k|}{|A_k|}>\rho$$

如果最后一式成立，则转入步骤（7），否则转入步骤（6）。

（6）取消识别结果，将输出层神经元 g 的输出值复位为 0，并将这一神经元排除在下次识别的范围之外，返回步骤（4）。当所有已利用过的神经元都无法满足步骤（5）中的最后一式时，则选择一个新的神经元作为分类结果，并进入步骤（7）。

（7）接受识别结果，并按照下式调整连接权值：

$$w_{ig}(t+1)=\frac{t_{gi}(t)a_i}{0.5+\sum_{i=1}^{N}t_{gi}(t)a_i}$$

$$t_{gi}(t+1)=t_{gi}(t)a_i$$

其中，$i=1,2,\cdots,N$。

（8）将步骤（6）中复位的所有神经元重新加入识别范围中，返回步骤（2）对下一个模式进行识别。

无论网络学习还是回想，都使用以上的规则。只不过在网络回想时，只对那些与未使



用过的输出神经元有关的连接权值向量 w_{ij} 和 t_{ij} 才进行初始化。其他连接权值向量保持网络学习后的值不变。当输入模式是一个网络已记忆的学习模式时，不需再进行网络权值的调整，这是因为当输入模式和网络记忆的学习模式完全相等，再按照权值调整公式进行调整时，网络连接权值不会发生任何变化。而当输入模式与网络记忆模式存在一定差异时，按照权值调整公式进行调整，将会影响网络原有模式的记忆效果。但是如果输入的是一个全新的模式，需要利用网络对其另加记忆时，则必须按照权值调整公式对网络连接权值进行调整。

尽管 ART-1 网络具有许多其他网络所没有的优点，但是它仅以输出层中某个神经元代表分类结果，而不是像 Hopfield 网络那样，把分类结果分散在各个神经元上来表示。所以，一旦输出层中某个输出神经元损坏，则会导致该神经元所代表类别的模式信息全部消失。这是 ART-1 网络一个很大的缺陷。

ART-2 型网络与 ART-1 型的主要区别是，ART-2 型网络以模拟量作为输入模式，同时在算法上做了一些相应的改进，并采用慢速学习方式，其抗干扰能力大大增强。ART-3 型网络是由多个 ART-1 型网络组成的复合阶层型网络。

5.3.3 ART-1 网络的应用实例

MATLAB 神经网络工具箱没有为 ART 型网络提供专门的函数，因此，利用现有的神经网络工具箱是无法实现 ART-1 网络的。但是，我们可以借助于 MATLAB 强大的数学计算功能来实现 ART-1 网络的训练和联想记忆功能。

现举一个简单的例子来演示利用 MATLAB 实现 ART-1 网络的过程。如图 5-9 所示，设 ART-1 网络有 5 个输入神经元和 20 个输出神经元。现有两组输入模式 A_1=(1,1,0,0,0)和 A_2=(1,0,0,0,1)，要求利用这两个模式来训练网络。根据上一节中的训练过程，该网络的训练步骤为下面几步。

（1）初始化。令 $w_{ij}=1/(N+1)=1/6$， $t_{ji}=1$，其中 $i=1,2,\cdots,5$， $j=1,2,\cdots,20$，令 $\rho=0.8$。

（2）将输入模式 A_1 提供给网络的输入层。

（3）求获胜的神经元。因为在网络的初始状态下，所有的前馈连接权 w_{ij} 均取相等的权值 1/6，所以各输入神经元均具有相同的输入加权和 s_j。这是可取任一个神经元作为 A_1 的分类代表，如第 1 个，令其输出值为 1。

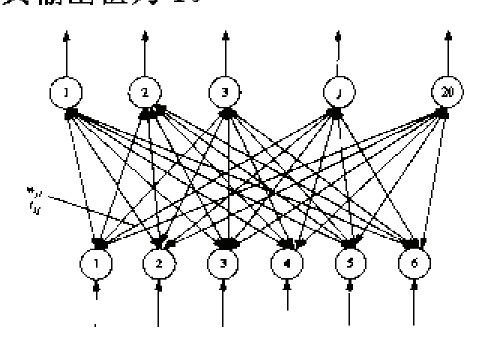


图 5-9 ART-1 网络实例

（4）计算下式：

$$|A_1| = \sum_{i=1}^{5} a_i = 2, \quad |T_1 A_1| = \sum_{i=1}^{5} t_{i1} a_i = 2$$

（5）计算 $\frac{|T_1 A_1|}{|A_1|} = 1 > 0.8$，接受这次识别结果。

（6）调整权值

$$W_1 = (w_{11}, w_{12}, w_{13}, w_{14}, w_{15}) = (0.4, 0.4, 0, 0, 0)$$
$$T_1 = (t_{11}, t_{21}, t_{31}, t_{41}, t_{51}) = (1, 1, 0, 0, 0)$$

至此，A_1 已经被记忆在网络中了。

（7）将输入模式 A_2 提供给网络的输入层。

（8）求获胜神经元，s_1=0.4，s_2=1/6，s_3=⋯=s_{20}=1/6，由于 $s_1 > s_2 = s_3 = \cdots = s_{20}$，所以取神经元1作为获胜神经元，但这显然与 A_1 的识别结果相矛盾。又因为

$$\frac{|T_2 A_2|}{|A_2|} = \frac{1}{2} < 0.8$$

所以拒绝这次识别结果，重新进行识别。由于 $s_2 = s_3 = \cdots = s_{20} = 1/6$，故可从中任选一个神经元作为 A_2 的分类结果，如神经元20。

（9）调整权值

$$W_2 = (w_{21}, w_{22}, w_{23}, w_{24}, w_{25}) = (0.4, 0, 0, 0, 0.4)$$
$$T_2 = (t_{12}, t_{22}, t_{32}, t_{42}, t_{52}) = (1, 0, 0, 0, 1)$$

至此，A_2 也记忆在网络中了。

按照上述步骤，可以编写以下MATLAB代码：

```
%竞争层的输出
jingzheng=rands(20)';
%正向权值W和反向权值T
W=rands(20,5);
T=rands(20,5);
%警戒参数
threshold=0.8;
%两组模式A1和A2
A1=[1 1 0 0 0];
A2=[1 0 0 0 1];
%初始化
for i=1:20
    for j=1:5
```

```
            W(i,j)=1/6;
            T(i,j)=1;
        end
    end
    %判定是否接受识别结果
    normalA1=norm(A1,1);
    normalTA1=T(1,:)*A1';
    count=1;
    if normalTA1/normalA1>threshold
        Jingzheng(count)=1;
    end
    %权值调整
    W(1,:)=[0.4 0.4 0 0 0 0];
    T(1,:)=[1 1 0 0 0];
    %寻找可以记忆 A2 的神经元
    for k=1:20
        s(k)=W(k,:)*A2';
        if s(k)==max(s)
            count=k;
        end
    end
    %如果和 A1 的神经元重复，继续寻找
    if Jingzheng(count)==1
        newcount=count+1
    end
    for i=1:(count-1)
        p(i)=s(i);
    end
    for i=count:19
        p(i)=s(i+1);
    end
    for k=newcount:20
        if s(k)==max(p)
            count=k;
        end
    end
    %确定找到的神经元序号 count，并令其对应的输出为 1
    Jingzheng(count)=1;
    %权值调整
    W(count,:)=[0.4,0,0,0,0,0.4];
    T(count,:)=[1,0,0,0,1];
    Jingzheng
```

运行结果为：

```
Jingzheng=
         1.0000   -0.2279   -0.7689    0.3715    0.3069   -0.6798    0.6467    0.6892
         0.8029    0.8101    0.9094   -0.9611    0.1225    0.8885   -0.0666    0.2544
```

-0.0220　0.9495　0.7052　1.0000

可见，第 1 个和第 20 个神经元的输出均为 1，表示它们记忆了输入模式。

5.4　学习矢量量化（LVQ）神经网络及 MATLAB 实现

学习矢量量化的英文全称是 Learning Vector Quantization，简称为 LVQ。LVQ 算法是在有教师状态下对竞争层进行训练的一种学习算法，它是从 Kohonen 竞争算法演化而来的。LVQ 神经网络在模式识别和优化领域有着广泛的应用。

5.4.1　LVQ 网络的结构

LVQ 网络结构如图 5-10 所示。一个学习矢量量化（LVQ）网络由三层神经元组成，即输入层、隐含层和输出层。该网络在输入层与隐含层间为完全连接，而在隐含层与输出层间为部分连接，每个输出神经元与隐含神经元的不同组相连接。隐含层和输出神经元之间的连接权值固定为 1。输入层和隐含神经元间的连接权值建立参考矢量的分量（对每个隐含神经元指定一个参考矢量）。在网络训练过程中，这些权值被修改。隐含神经元（又称为 Kohonen 神经元）和输出神经元都具有二进制输出值。当某个输入模式被送至网络时，参考矢量最接近输入模式的隐含神经元因获得激发而赢得竞争，因而允许它产生一个“1”。其他隐含神经元都被迫产生“0”。与包含获胜神经元的隐含神经元组相连接的输出神经元也发出“1”，而其他输出神经元均发出“0”。产生“1”的输出神经元给出输入模式的类，每个输出神经元被表示为不同的类。



图 5-10　LVQ 网络结构

5.4.2 LVQ网络的学习规则

在介绍网络学习规则之前，首先定义一个变量。假定网络输入层的输入向量为 $X=(x_1,x_2,\cdots,x_M)$，其中，M 为输入层神经元的数目；输入层和竞争层之间的连接权值矩阵为$W^1=(w_1^1,w_2^1,\cdots,w_p^1)$，$w_i^1=(w_{i1}^1,w_{i2}^1,\cdots,w_{iM}^1)$。$w_{ij}^1$中，$i=1,2,\cdots,p$，$j=1,2,\cdots,M$，表示输入层第 i 个神经元和第 j 个神经元之间的连接权值，p 为竞争层神经元的数目，竞争层的输出向量为 $V=(v_1,v_2,\cdots,v_p)$，竞争层与输出层神经元之间的连接权值矩阵为 $W^2=(w_1^2,w_2^2,\cdots,w_N^2)$，其中 $w_k^2=(w_{k1}^2,w_{k2}^2,\cdots,w_{kp}^2)$。$w_{kr}^2$，$k=1,2,\cdots,N$，$r=1,2,\cdots,p$，表示竞争层第 k 个神经元与输出层第 r 个神经元之间的连接权值，N 为输出层神经元的数目。竞争层的每个神经元通过学习原型向量，并对输入空间进行分类。将竞争层学习得到的类称为子类，将输出层学习得到的类称为目标类。

LVQ 神经网络的学习规则结合了竞争学习和有教师学习的规则，即需要一组正确网络行为的例子来训练该网络。假定有以下的训练模式：

$$\{x_1,t_1\},\{x_2,t_2\},\cdots,\{x_Q,t_Q\}$$

其中，每个目标输出向量t_j（$j=1,2,\cdots,Q$） 有且只有一个分量为 1，其他分量全都为 0。

为了使得学习过程可以进行下去，通常把第一层的每个神经元指定给一个输出神经元，这样就可以定义矩阵W^2。W^2的列表示子类，行表示类。W^2的每一列仅有一个 1 出现的行，表明这个子类属于该行表示的类，即

$$W_{kr}^2=\begin{cases}1 & \text{有且仅有 } r\in k \\ 0 & \text{如果} r\in k\end{cases}$$

W^2一旦定义好就不再改变，神经网络的学习是通过改进了的 Kohonen 规则使W^1改变来进行的，即在每次迭代过程中，将一个输入向量 X 提供给网络，并且通过竞争层计算每个原型向量与 X 之间的距离，与 X 的距离最近的神经元i^*获得竞争胜利，输出 V 的第i^*个元素值设定为 1，通过下式可计算得到输出向量 Y 的值：

$$Y=W^2V$$

显而易见，Y 中的每个分量也只有一个非零元素。假定其序号为k^*，这样一来，就表明 X 是指定给k^*类的。

下面对分类结果进行分析。第一，如果分类正确，即若 $y_{k^*}=t_{k^*}=1$，则获胜的隐含神

经元将沿着 X 的方向移动，按照下式修正竞争层权值向量：

$$i^{*w}(t+1)=i^{*w}(t)+\eta\left(p(t+1)-i^{*w}(t)\right)$$

第二，如果分类不正确，即若 $y_{k^*}=1$，而 $t_{k^*}=0$ 时，表示错误的隐含层神经元竞争获胜，则移动该神经元的权值远离 X，修改竞争层权值向量：

$$i^{*w}(t+1)=i^{*w}(t)-\eta\left(p(t+1)-i^{*w}(t)\right)$$

其中，$\eta\in(0,1)$ 为比例系数，在调整权值矩阵过程中反映调整速率。$i^{*w}(t)$ 表示竞争层中第 i^* 个神经元在 t 时刻的权值。经过这样的处理后，每个神经元移向那些落入形成子类的类中的向量，而远离那些落入其他类中的向量。

5.4.3 基于 LVQ 网络的模式识别

神经网络通过寻找输入/输出数据之间的关系，来实现特征提取和统计分类等模式识别任务。经过几十年的发展，神经网络已经奠定了在模式识别领域中不可或缺的位置。在模式识别的应用中，单层感知器是能一致逼近线性连续函数空间最简单的神经网络，但是它具有其固有的局限性，即对非线性样本空间不可分。BP 网络是一种应用最为普遍的网络，其缺点在于采用了基于梯度下降的非线性优化策略，有可能陷入局部最小问题，不能保证求出全局最小值。其他一些优化策略如遗传算法、模拟退火算法等，虽然可以求得全局最小，但是计算量很大，有时候会出现效率问题。这里尝试利用 LVQ 网络来实现模式识别，LVQ 网络的优点是不需要将输入向量进行归一化、正交化，只需要直接计算输入向量与竞争层之间的距离，从而实现模式识别，因此简单易行。

下面给出一组输入向量，设计一个 LVQ 网络，根据给定的目标，将输入向量进行模式识别。

1. 输入向量的目标向量的设计

设输入为一个二维向量，一共有 12 个点。

```
P=[-6 -4 -2 0 0 0 0 2 4 6;0 2 -2 1 2 -2 1 2 -2 0];
```

这些样本的类别为：

```
C=[1 1 1 2 2 2 2 1 1 1]
```

这表明，前 3 组样本和后 3 组样本属于第一类，中间 4 组样本属于第二类。由此可见，分类的依据是：（a,b）属于第一类，a、b 都是任意实数，其中 a≠0；（0,b）属于第二类。

将类别向量 C 利用函数 vec2ind 转换为网络可以使用的目标向量 T：

```
T=ind2vec(C)
```

将这些向量绘制在一张图上，如图 5-11 所示。

```
plotvec(P,C,'*');
axis([-8 8 -3 3]);
```



图 5-11　输入向量的分布

函数 plotvec 可以根据类别向量的情况，利用不同的颜色在一张图上绘制出输入向量 P 中的相应元素。其中，两边的星号表示第一类的数据，中间的星号表示第二类的数据。

2. 网络创建与训练

神经网络工具箱中用于创建 LVQ 网络的函数为 newlvq，利用该函数创建一个 LVQ 网络。

```
net = newlvq(minmax(P),5,[0.6 .04]);
```

其中，网络中间层的神经元个数设定为 5 个。[0.6 0.4]表示在中间层的权值中，有 60%的列的第一行的值为 1，40%的列的第一行值为 1。也就是说，有 60%的列属于第一类，40%的列属于第二类。学习速率和学习函数都采用默认值，分别为 0.01 和 leranlv1。

检查该网络的初始权值，并将其与输入向量绘制在一张图上，如图 5-12 所示。

```
w1=net.IW{1};
plot(w1(1,1),w1(1,2),'+r');
```

结果为：

```
w1=
    0    0
    0    0
    0    0
    0    0
    0    0
```

可见，由于没有经过训练，网络的初始权值都为零。所以，在图 5-12 中重合为一个“+”号。

接下来通过函数 train 对网络进行训练，训练步数设为 100。

```
net.trainParam.epochs=100;
net=init(net);
net=train(net,P,T);
```

训练结果为：

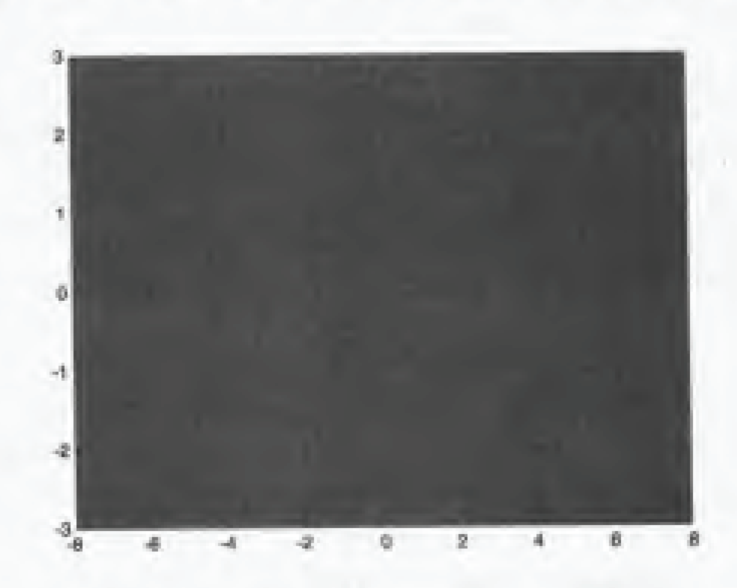


图 5-12 网络输入向量和初始权值向量

```
TRAINR, Epoch 0/100
TRAINR, Epoch 3/100
TRAINR, Performance goal met.
```

可见，网络的训练函数为 trainr，经过 3 次训练后，网络误差达到要求，如图 5-13 所示。

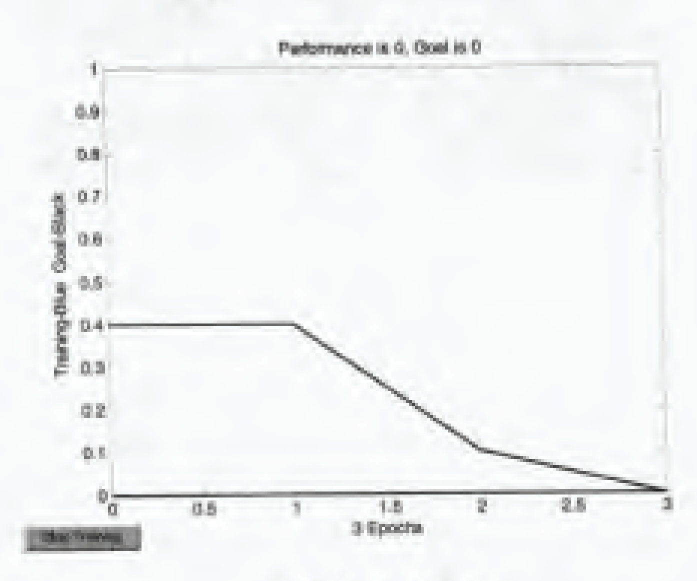


图 5-13 训练结果

再次检查网络竞争层的权值，并将其与输入向量绘制在一张图上。如图 5-14 所示，由图可见，网络经过训练后，竞争层神经元的权值发生了变化，显然，这种变化有助于对两类模式进行分类。

```
plotvec(P,C,'*r');
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'+');
axis([-8 8 -3 3]);
```

对于训练好的网络来说，其权值是固定的。对于每个输入值，网络都会提供相应的分类。因此，可以利用这一点对网络进行测试。

设置两个特定的点，分别属于第一类和第二类。（0 0.2）属于第二类，（1 0）属于第一类。其中利用仿真函数 sim 来得到网络的输出，在命令行窗口中输入以下代码后按回车键：

```
p=[0 1;0.2 0];
y=sim(net,p);
yc=vec2ind(y)
```

输出结果为：

```
yc =
      2     1
```

这说明（0 0.2）属于第二类，（1 0）属于第一类，和实际情况是相符合的。由此可知网络的分类性能是很好的。

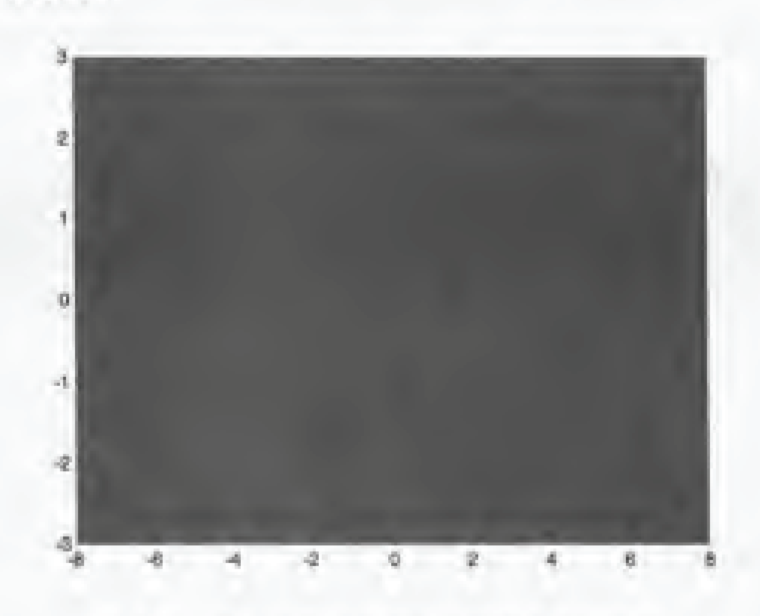


图 5-14　输入向量及训练后的网络竞争层权值

以上表明，利用 LVQ 网络进行模式识别是合适的。与其他的模式识别或映射方式相比较，采用此网络的优点在于，网络结构简单，而且它只通过内部单元的相互作用，就可以完成十分复杂的分类处理，也很容易将设计域中各种繁杂分散的设计条件收敛到结论上来。因此，可以说 LVQ 网络具有很好的模式分类识别特性。在这个过程中设计人员不需要构造复杂的，甚至是难以构造的非线性处理函数。此外，LVQ 网络表现出比 BP 网络和 ART 网络更强的容错性和鲁棒性，不易导致系统的崩溃。一般设计者只要使用已建立的网络分类决策系统，输入实际所需的性能指标就能得到较为满意的识别结果。

本实例的 MATLAB 代码为：

```
%输入向量 P 及其类别 C
P=[-6 -4 -2  0 0 0 0 2 4 6;0 2 -2 1 2 -2 1 2 -2 0];
C=[1 1 1 2 2 2 2 1 1 1];
%将类别向量 C 转换为目标向量 T
T=ind2vec(C);
%绘制输入向量 P
```

```
plotvec(P,C,'*r');
axis([-8 8 -3 3]);
hold on;
%网络创建
%竞争层有5个神经元
net = newlvq(minmax(P),5,[0.6 0.4]);
%求网络竞争层的初始权值w1
w1=net.IW{1};
w1
%将输入向量和权值向量绘制在一张图上
plot(w1(1,1),w1(1,2),'+r');
axis([-8 8 -3 3]);
hold off;
%网络训练
%训练步数为100
net.trainParam.epochs=100;
net=train(net,P,T);
%将输入向量和训练后的权值向量绘制在一张图上
plotvec(P,C,'*r');
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'+');
axis([-8 8 -3 3]);
hold off;
%在两类中分别指定两个点，测试网络的性能
p=[0 1;0.2 0];
y=sim(net,p);
yc=vec2ind(y)
yc
```

5.5 对向传播网络（CPN）及MATLAB实现

对向传播（Counter Propagation）网络，简称CPN，是将Kohonen特征映射网络与Grossberg基本竞争型网络相结合，发挥各自特长的一种新型特征映射网络。这一网络是美国计算机专家Robert Hecht-Nielsen于1987年提出的。这种网络被广泛地应用于模式分类、函数近似、统计分析和数据压缩等领域。

5.5.1 CPN概述

CPN网络结构如图5-15所示。由图可见，网络分为输入层、竞争层和输出层。输入层与竞争层构成SOM网络，竞争层与输出层构成基本竞争型网络。从整体上看，网络属于有教师型的网络，而由输入层和竞争层构成的SOM网络又是一种典型的无教师型神经网络。因此，这一网络既汲取了无教师型网络分类灵活、算法简练的优点，又采纳了有教师型网络分类精细、准确的长处，使两种不同类型的网络有机地结合起来。



CPN 的基本思想是，由输入层至输出层，网络按照 SOM 学习规则产生竞争层的获胜神经元，并按这一规则调整相应的输入层至竞争层的连接权；由竞争层到输出层，网络按照基本竞争型网络学习规则，得到各输出神经元的实际输出值，并按照有教师型的误差校正方法，修正由竞争层到输出层的连接权。经过这样的反复学习，可以将任意的输入模式映射为输出模式。

从这一基本思想可以发现，处于网络中间位置的竞争层获胜神经元及与其相关的连接权向量，既反映了输入模式的统计特性，又反映了输出模式的统计特性。因此，可以认为，输入、输出模式通过竞争层实现了相互映射，即网络具有双向记忆的功能。如果输入/输出采用相同的模式对网络进行训练，则由输入模式至竞争层的映射可以认为是对输入模式的压缩；而由竞争层至输出层的映射可以认为是对输入模式的复原。利用这一特性，可以有效地解决图像处理及通信中的数据压缩及复原问题，并可得到较高的压缩比。

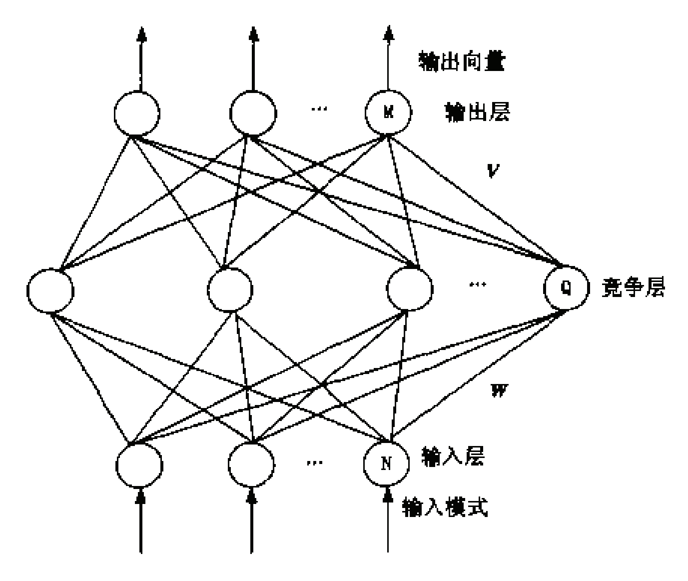


图 5-15　CPN 网络结构

接下来介绍 CPN 的学习及工作规则。假定输入层有 N 个神经元，p 个连续值的输入模式为 $A_k=\left(a_1^k,a_2^k,\cdots,a_N^k\right)$，竞争层有 Q 个神经元，对应的二值输出向量为 $B_k=\left(b_1^k,b_2^k,\cdots,b_Q^k\right)$，输出层有 M 个神经元，其连续值的输出向量为 $C_k'=\left(c_1'^k,c_2'^k,\cdots,c_M'^k\right)$，目标输出向量为 $C_k=\left(c_1^k,c_2^k,\cdots,c_M^k\right)$，以上 $k=1,2,\cdots,p$。由输入层至竞争层的连接权值向量为 $W_j=\left(w_{j1},w_{j2},\cdots,w_{jN}\right)$，$j=1,2,\cdots,Q$；由竞争层到输出层的连接权向量为 $V_l=\left(v_{l1},v_{l2},\cdots,v_{lQ}\right)$，$l=1,2,\cdots,M$。网络学习和工作规则如下所述。

（1）初始化。将连接权向量 W_j 和 V_l 赋予区间[0,1]内的随机值。将所有的输入模式 A_k 进行归一化处理：

$$a_i^k = \frac{a_i^k}{\| A_k \|}, \| A_k \| = \sqrt{\sum_{i=1}^{N} \left(a_i^k\right)^2}, i = 1,2,\cdots,N$$

（2）将第 k 个输入模式 A_k 提供给网络的输入层。

（3）将连接权值向量 W_j 按照下式进行归一化处理：

$$w_{ji} = \frac{w_{ji}}{\| w_{ji} \|}, \quad \| w_{ji} \| = \sqrt{\sum_{i=1}^{N} w_{ji}^2} \qquad i = 1,2,\cdots,N$$

（4）求竞争层中每个神经元的加权输入和：

$$s_j = \sum_{i=1}^{N} a_i^k w_{ji} \qquad j = 1,2,\cdots,Q$$

（5）求连接权向量 W_j 中与 A_k 距离最近的向量 W_g：

$$W_g = \max_{j=1,2,\cdots,Q} \sum_{i=1}^{N} a_j^k w_{ji} = \max_{j=1,2,\cdots,Q} s_j$$

将神经元 g 的输出设定为 1，其余竞争层神经元的输出设定为 0：

$$b_j = \begin{cases} 1 & j = g \\ 0 & j \neq g \end{cases}$$

（6）将连接权向量 W_g 按照下式进行修正：

$$w_{gi}(t+1) = w_{gi}(t) + \alpha\left(a_i^k - w_{gi}(t)\right) \qquad i = 1,2,\cdots,N$$

其中，$1 < \alpha < 1$ 为学习率。

（7）将连接权向量 W_g 重新归一化，归一化算法同上。

（8）按照下式修正竞争层到输出层的连接权向量 V_l：

$$v_{lj}(t+1) = v_{lj}(t) + \beta b_j \left(c_l - c_l^{'}\right) \qquad l = 1,2,\cdots,M, \quad j = 1,2,\cdots,Q$$

其中，$1 < \beta < 1$ 为学习率。由步骤（5）可将上式简化为：

$$v_{lg}(t+1) = v_{lg}(t) + \beta b_j \left(c_l - c_l^{'}\right) \qquad l = 1,2,\cdots,M$$

由此可见，只需调整竞争层获胜神经元 g 到输出层神经元的连接权向量 V_g，其他连接权向量保持不变。

（9）求输出层各神经元的加权输入，并将其作为输出神经元的实际输出值，$c_l' = \sum_{j=1}^{g} b_j v_{lg}$，$l = 1,2,\cdots,M$，同理可将其简化为 $c_l' = v_{lg}$。

（10）返回步骤（2），直到将 p 个输入模式全部提供给网络。

（11）令 $t = t + 1$，将输入模式 A_k 重新提供给网络学习，直到 $t = T$。其中 T 为预先设定的学习总次数，一般取 $500 < T < 10000$。

5.5.2 CPN应用实例

这里举一个非常简单而且和日常生活相关的例子来说明CPN网络的应用。现在需要创建一个CPN网络，其任务是在已知一个人本星期应该完成的工作量和此人当时的思想情况状态的情况下，对此人星期日下午的活动安排提出建议。

按照一般情况，将工作量分为3个档次，即没有、有一些和很多，所对应的量化值分别为0.0、0.5和1.0；把思想情绪也分为3个水平，即低、一般和高，所对应的量化值分别为0.0、0.5和1.0。可选择的活动有5个，即在家里看画报、去商场购物、到公园散步、与朋友一起吃饭和干工作。工作量和思想情绪状态一共有6种组合，这6种组合分别对应各自的最佳活动选择。样本模式如表5-6所示。

表5-6　网络训练样本模式

工作量	思想情绪	活动安排	目标输出
没有 0.0	低 0.0	看画报	10000
有一些 0.5	低 0.0	看画报	10000
没有 0.0	一般 0.5	购物	01000
很多 1.0	高 1.0	公园散步	00100
有一些 0.5	高 1.0	吃饭	00010
很多 1.0	一般 0.5	工作	00001

把这组训练样本提供给网络进行充分学习后，网络就具有了一种“内插”功能，即当网络输入一对在（0,1）区间中反映工作量和情绪的量化值后，网络将自动根据原有的记忆，找出对应于这对量化值的最佳活动选择，以输出模式的形式提供给用户作为决策参数。

实际上，不光CPN网络具有这种“内插”功能，BP网络、SOM网络都具有这种功能。从模式识别的角度上讲，这些网络具有对输入模式进行分类的功能。

可惜的是，对功能如此强大的CPN网络，神经网络工具箱中竟然没有为之支持的函数工具。但是，既然上一节中已经给出了有关CPN的学习和训练算法过程，因此，我们可以

利用 MATLAB 强大的数学计算功能，实现解决该问题的 CPN 网络。

根据题意，该网络的输入层应该有 2 个神经元，输出层应该有 5 个神经元。为了更加准确地解决问题，将竞争层神经元设置为 18 个。网络结构如图 5-16 所示。

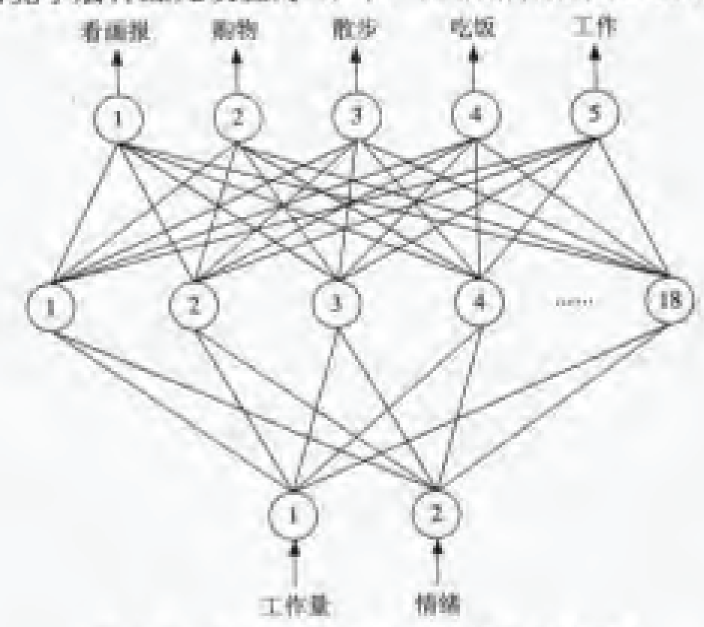


图 5-16 星期日下午活动安排决策 CPN 网络

由表 5-6 可得，网络的输入向量为：

```
P=[0 0;0.5 0.5; 0 0.5;1 1; 0.5 1;1 0.5]'.
```

目标向量为：

```
T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1]'.
```

下面对网络进行一个周期的学习。令输入层和竞争层之间的连接权向量矩阵用 W 表示，竞争层和输出层之间的权向量矩阵用 V 表示。可知 W 为一个 18×2 的矩阵，V 是一个 5×18 的矩阵。学习速率设定为 0.1。

（1）初始化。利用 MATLAB 中的随机数产生函数 W 和 V，并赋以区间[0,1]之间的随机值，代码为：

```
w=rands(18,2)/2+0.5;
v=rands(5,18)/2+0.5;
```

由于函数 rands 产生的随机数位于区间(-1,1)之间，所以这里做了这样的处理，使得产生的随机数既不影响随机性能，又位于区间[0 1]中。

对输入向量进行归一化处理：

```
for i=1:6
    if(P(i,:)==[0 0])
        P(i,:)=P(i,:)
    else
        P(i,:)=P(i,:)/norm(P(i,:));
    end
end
```

之所以要在循环中进行数据判断，是因为向量[0 0]是无法归一化处理的，比如 P 的第一组元素，就是正在用的这一组中。

（2）将第一个输入样本(0 0)提供给网络的输入层神经元。

（3）对连接权向量 W 进行归一化处理。

（4）求每一个竞争层神经元的加权输入 s_j，j=1,2,…,18：

```
for i=1:18
    w(i,:)=w(i,:)/norm(w(i,:));
    s(i)=P(1,:)*w(i,:)';
end
```

循环语句中第 1 句用于对连接权向量 W 进行归一化处理，第 2 句可以求出竞争层每个神经元的输出。结果为：

```
s= 0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0
```

（5）求连接权向量 W 中与(0 0)距离最近的向量，由于输出全部为 0，所以可任选一个权值向量 Wg，这里选为 18，并将该神经元的输出设定为 1。

```
for i=1:18
    w(i,:)=w(i,:)/norm(w(i,:));
    s(i)=P(j,:)*w(i,:)';
end
temp=max(s);
for i=1:18
    if temp==s(i)
        count=i;
    end
end
%将所有竞争层神经元的输出置 0
for i=1:18
    s(i)=0;
end
%选中的神经元输出为 1
s(count)=1;
```

（6）调整连接权向量 W18，并重新将其归一化

```
w(count,:)=w(count,:)+0.1*[P(1,:)-w(count,:)];
w(count,:)=w(count,:)/norm(w(count,:))
```

输出结果为：

```
w(18,:)=
         0.9758    0.2186
```

经检验，此时的 W18 确实已经归一化了。

（7）调整竞争层神经元到输出层神经元之间的连接权向量 V：

```
v(:,count)=v(:,count)+0.1*(T(1,:)'-T_out(1,:)');
```

由于此时输出层神经元的输出域目标向量是一致的，所以这里的权值是没有经过调整的。等到来了下一个模式后，权值才会真正得到调整。

（8）计算输出层各神经元的加权输入，并将其作为神经元的实际输出值：

```
T_out(1,:)=v(:,count)';
```

第一组实际输出就等于竞争层中第 18 个神经元与输出层各神经元之间调整后的连接

权值。

（9）返回步骤（2），将输入向量中的(0.5 0.5)提供给网络。

（10）继续学习，直到训练次数达到设定的最大值。

CPN网络训练结束后，按照以下步骤进行网络回想：

（1）将输入模式A提供给网络的输入层。

（2）根据下式求出竞争层的获胜神经元 g：

$$b_g = \max_{i=1,2,\cdots,Q}\left(\sum_{i=1}^{N} w_{ji} a_i\right)$$

（3）令 b_g=1，其余的输出都等于0。按照下式求得输出层各神经元的输出：

$$c_j = v_{jg} b_g$$

由此产生了输出模式 $C=(c_1,c_2,\cdots,c_M)$，从而就得到了输入A的分类结果。

根据以上步骤，可编写出以下MATLAB代码用于网络训练并回想：

```
%初始化正向权值w和反向权值v
w=rands(18,2)/2+0.5;
v=rands(5,18)/2+0.5;
%输入向量P和目标向量T
P=[0 0;0.5 0.5;0 0.5;1 3;0.5 1;1 0.5];
T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1];
T_out=T;
%设定学习步数为1000次
epoch=1000;
%归一化输入向量P
for i=1:6
    if P(i,:)==[0 0]
        P(i,:)=P(i,:);
    else
        P(i,:)=P(i,:)/norm(P(i,:));
    end
end
%开始训练
while epoch>0
for j=1:6
    %归一化正向权值w
    for i=1:18
        w(i,:)=w(i,:)/norm(w(i,:));
        s(i)=P(j,:)*w(i,:)';
    end
    %求输出为最大的神经元，即获胜神经元
    temp=max(s);
    for i=1:18
        if temp==s(i)
            count=i;
```

```
            end
        end
        %将所有竞争层神经元的输出置0
        for i=1:18
            s(i)=0;
        end
        %将获胜神经元的输出置1
        s(count)=1;
        %权值调整
        w(count,:)=w(count,:)+0.1*[P(j,:)-w(count,:)];
        w(count,:)=w(count,:)/norm(w(count,:));
        v(:,count)=v(:,count)+0.1*(T(j,:)'-T_out(j,:)');
        %计算网络输出
        T_out(j,:)=v(:,count)';
    end
        %训练次数递减
        epoch=epoch-1;
end
%训练结束
T_out
%网络回想
%网络的输入模式 Pc
Pc=[0.5 1;1 3];
%初始化 Pc
for i=1:2
    if Pc(i,:)==[0 0]
        Pc(i,:)=Pc(i,:);
    else
        Pc(i,:)=Pc(i,:)/norm(Pc(i,:));
    end
end
%网络输出
Outc=[0 0 0 0 0;0 0 0 0 0];
for j=1:2
    for i=1:18
        sc(i)=Pc(j,:)*w(i,:)';
    end
    tempc=max(sc);
    for i=1:18
        if tempc==sc(i)
            countp=i;
        end
        sc(i)=0;
    end
    sc(countp)=1;
    Outc(j,:)= v(:,countp)';
```

```
end
%回想结束
Outc
```

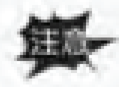
输出结果为：

```
T_out =
        1.0000    0.0000    0.0000    0.0000    0.0000
        1.0000    0.0000    0.0000    0.0000    0.0000
        0.0000    1.0000    0.0000    0.0000    0.0000
        0.0000    0.0000    1.0000    0.0000    0.0000
        0.0000    0.0000    0.0000    1.0000    0.0000
        0.0000    0.0000    0.0000    0.0000    1.0000
```

由此可见，经过1000次训练后，网络的实际输出和目标输出就一致了，这说明训练过程是有效的。

```
Outc =
        0.0000    0.0000    0.0000    1.0000    0.0000
        0.0000    0.0000    1.0000    0.0000    0.0000
```

Outc是网络回想的输出，实际上也就是网络测试的结果，在这里给出了两种特定的组合状态，即(0.5 1)和(1 1)的组合，这两种组合分别对应吃饭和到公园散步，可见，网络给出了正确的建议。



在以上代码中，训练输入向量P和回想输入向量Pc中的(1 1)都被(1 3)所替代，这是因为经过归一化处理后，P中两个本来不一致的样本(0.5 0.5)和(1 1)变得一致了，而它们对应的输出向量却并不一致，因此，将其用(1 3)就是为了避免这种情况，对结果并没有影响。

5.6 小　　结

本章主要介绍了自组织竞争网络、特征映射网络、自适应共振网络、LVQ神经网络和CPN网络共五种自组织神经网络模型及其实现过程。这些神经网络都采用无教师的学习方式，在模式聚类领域有着比较广泛的应用。神经网络工具箱为自组织竞争网络、特征映射网络和LVQ网络提供了函数，利用这些网络进行模式聚类时需要注意以下几个问题：

（1）根据实际问题为竞争层选择合理的神经元个数，并进行恰当的排列。

（2）一般来说，训练次数和训练速率都会影响到分类精度，综合考虑分类精度和运算时间来确定合适的训练次数和训练速率。一般是首先选择一个比较大的训练次数和比较小的训练速率，不断观察分类结果，当分类结果不再发生改变时，对应的训练次数就是合理的次数。



第 6 章　图形用户界面 GUI

所谓图形用户界面（Graphical User Interfaces，GUI），指的是由窗口、光标、按键、菜单、文字说明等对象构成的一个用户界面。用户可以通过一定的方法（如鼠标或键盘）选择、激活这些图形对象，实现某种特定的功能，如计算、绘图等，能够在很大程度上提高工作效率。

MATLAB 神经网络工具箱从 4.0 版本开始提供 GUI，经过不断改进与完善，目前的 NN Toolbox 4.0.3 已经非常容易使用了，而且功能也比较强大。本章将重点介绍使用最新版本的 GUI 如何进行神经网络的设计、分析及应用。

本章主要内容有：

- 网络设计
- 网络训练与仿真
- 数据操作

6.1　概　　述

GUI 的设计是为了使用户可以更友好和更方便地使用 MATLAB 的神经网络工具箱。GUI 是一个独立的窗口，即 Network/Data Manager 窗口，如图 6-1 所示。这个窗口相对于 MATLAB 命令行窗口是独立的，但可以将 GUI 得到的结果数据导出到命令窗口中，也可以将命令窗口中的数据导入到 GUI 窗口中。



图 6-1　Network/Data Manager 窗口样本向量

GUI 开始运行后，我们就可以创建一个神经网络，而且可以查看其结构，对其进行仿真和训练，也可以输入和输出数据。

接下来，我们通过 GUI 来创建一个网络，演示 GUI 的使用流程和应用技巧。

6.2 网络设计

本节将通过 GUI 设计一个前向 BP 网络，对正弦信号进行逼近。该 BP 网络的中间层神经元数目为 10 个，输入层神经元的传递函数为 S 型正切函数，中间层神经元的传递函数为纯线性函数，网络训练函数设定为 trainlm，待近似的原始正弦信号为 $f(t)=\sin(\pi t)$。

下面开始创建网络。

首先，在 MATLAB 命令行窗口中输入 nntool 后按回车键，会出现如图 6-1 所示的 Network/Data Manager 窗口。由图可见，该窗口可以分为 3 个部分，最上面的区域由一些空白的文本框组成，中间和底部的区域都是由各类按钮组成的。

接下来，我们分别介绍它们的作用。首先，我们需要创建一个 BP 网络。单击【New Network】按钮可以出现如图 6-2 所示的窗口。在 Network Type 的下拉框中选择需要创建的 BP 网络类型，为 Feed-forward backprop。在 Network Name 文本框中键入所创建的网络的名称，这里将该网络命名为 simubp。然后，在 Input Range 文本框中设定输入向量元素的最大值和最小值的范围，这里输入的范围为区间[-1 1]。在它的左边有一个下拉框，这是设定输入向量的来源，这里我们取默认值，不对其做出任何改动。Training function 下拉框用于选择训练函数的类型，这里选择 TRAINLM。Performance function 取默认值为 MSE。Number of layers 用于设定网络的层数，这里键入 2，表明我们所设计的网络有一层是中间层。最后设置网络各层的传递函数和神经元的数目，在 Properties for 下拉框中选择 Layer 1，表示接下来将设置网络中间层的传递函数和神经元的数目。在 Number of neurons 文本框中键入 10，表示中间层由 10 个神经元组成，在 Transfer Function 下拉框选择传递函数类型为 TANSIG。然后在 Properties for 下拉框中选择 Layer 2，表示接下来对输出层进行属性设置。在 Number of neurons 文本框中键入 1，表示输出层一共有 1 个神经元，在 Transfer Function 下拉框选择为 PURELIN。此时，单击【View】按钮，就可以看到刚刚设计好的 BP 网络，如图 6-3 所示。单击【Create】按钮，就可以将设计好的网络 simubp 保存在 MATLAB 工作空间中。

让我们再回到 Network/Data Manager 窗口，单击【New Data】按钮，出现如图 6-4 所示的窗口。窗口右侧的 Name 文本框用于设定数据向量的名称，这里键入 p。Value 文本框用于接收数据向量，数据向量可以是直接的向量形式，如[1 2 3; 4 5 6]；也可以是间接的 MATLAB 数学表达式，这里采用的是后者，为“-1:0.05:1”。对话框右侧用于设定数据类型，即提供的数据向量是作为网络的输入向量还是目标向量等。这里的 p 表示输入向量，最后单击【Create】按钮，就为网络创建了一个名称为 p 的输入向量。按照同样的方式，可以设定网络的目标向量 t=sin(pi*p)。

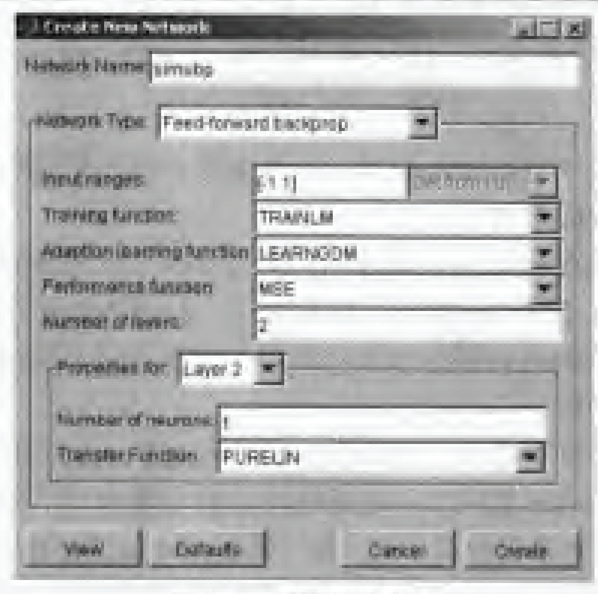


图 6-2　网络创建窗口

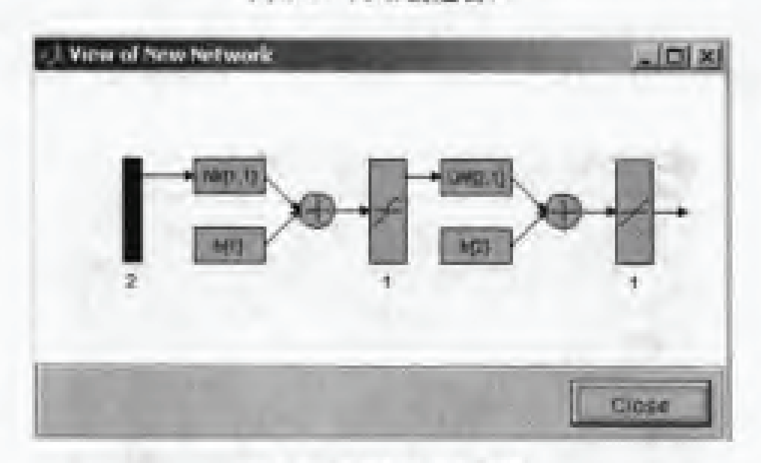


图 6-3　创建的 BP 网络

这样一来，我们就通过 GUI 创建了一个名为 simubp 的前向 BP 网络，该网络的输入向量为 p，目标向量为 t，其目的是为了对一个标准正弦信号进行模拟。

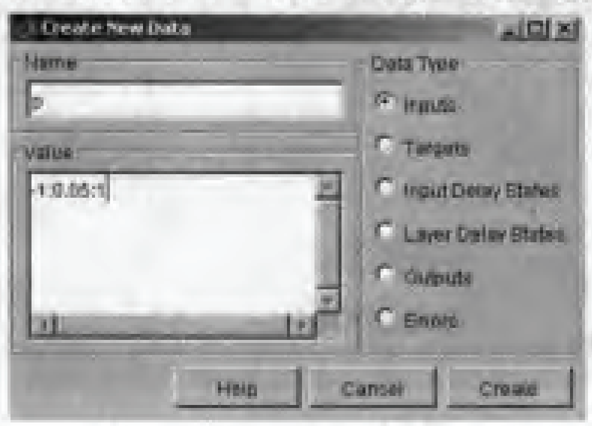


图 6-4　创建输入和目标数据向量等

6.3 网络训练与仿真

网络需要经过训练后才可以投入使用，本节主要介绍如何利用 GUI 对网络进行训练，并通过仿真来检验网络的应用情况。

再次回到 Network/Data Manager 窗口，在 Networks 文本框中选中 simubp，位于窗口底部的四个按钮【Initialize...】、【Simulate...】、【Train...】和【Adapt...】被激活。单击【Train...】按钮，出现如图 6-5 所示的对话框，该对话框用于设置网络自适应、仿真、训练和初始化的参数，并执行相应的过程。

为了说明训练前的网络不适于实际应用，在训练之前，我们首先对网络进行一次仿真。在以上对话框中选中 Simulate 选项卡，激活网络仿真设置窗口，如图 6-6 所示。

按照图 6-6 所示的方式设置仿真参数，将右侧的 Simualtion Results-Outputs 文本框内容修改为 simubp_sim，用于表示网络的仿真输出。单击右下角的【Simulate Network】按钮，可得出网络的仿真输出 simubp_sim，将其输出到 MATLAB 的工作空间中（具体方式在下节中介绍），网络的逼近结果如图 6-7 所示。由图可见，训练前的网络对于信号的模拟效果是非常差的。

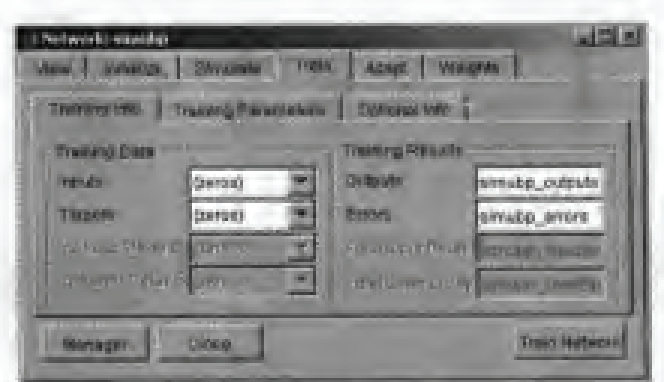


图 6-5 网络训练对话框

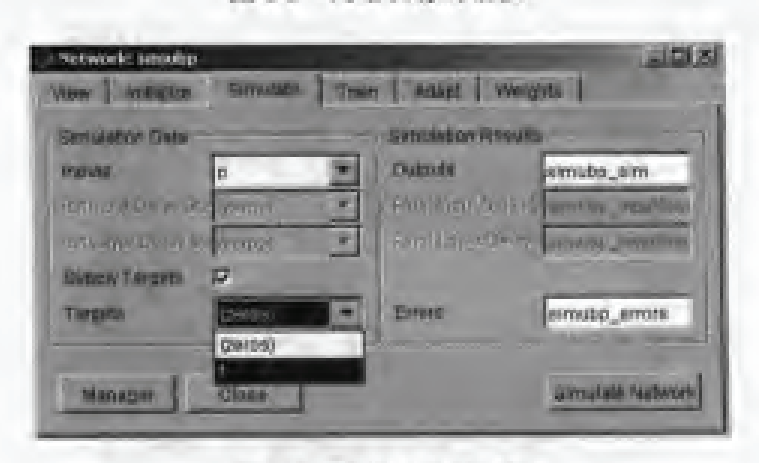


图 6-6 网络仿真对话框

接下来设定训练参数，在 Training info-Training Data-Inputs 下拉框中选择 p 作为输入向量，在 Training Data-Targets 下拉框中选择 t 作为目标向量，其余的项目不做改动。然后选中 Training Parameters 选项卡，此选项卡用于设定网络的训练参数。这里将训练步数 epochs

设置为50次，训练目标goal设置为0.01，其余的均取默认值，如图6-8所示。

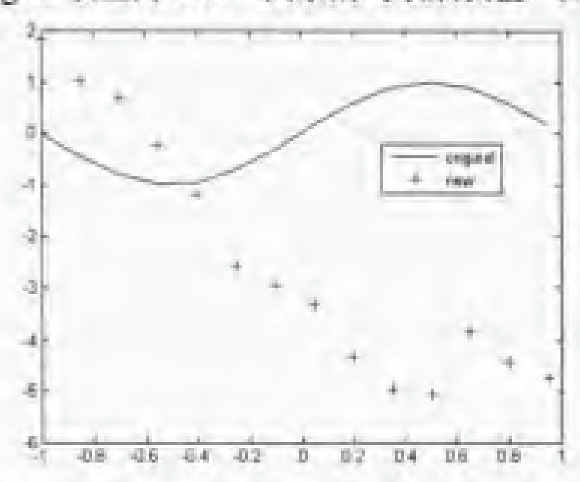


图6-7 训练前的网络逼近结果

图6-8 训练参数设定

此外，参数设定页面中还有一个Optional Info选项卡，该选项卡主要用于设定确认向量和检验向量，本例不需要设定这些参数。

单击右下角的【Train Network】按钮，开始训练网络。训练结果如图6-9所示，由图可见，网络的训练过程收敛得很快，经过两步后，网络的目标就已经达到要求。

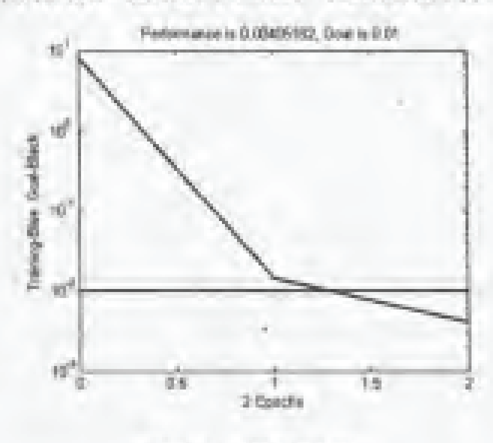


图6-9 训练结果

通过仿真可以检验训练后的网络对信号的逼近效果。通过以下的代码可以将网络得到的仿真输出和实际信号的输出绘制在一张图上，如图6-10所示，这样就可以很直观地检验网络对信号的逼近效果。可以看出，训练后的网络对正弦信号的逼近效果是非常好的。

```
plot(p,t,'b-');
```

```
hold on
plot(p,simubp_sim,'r+')
```



仿真时，将仿真输出向量的名称设定为 simubp_sim，其目的是为了避免和训练时的输出向量 simubp_outputs 相混淆。

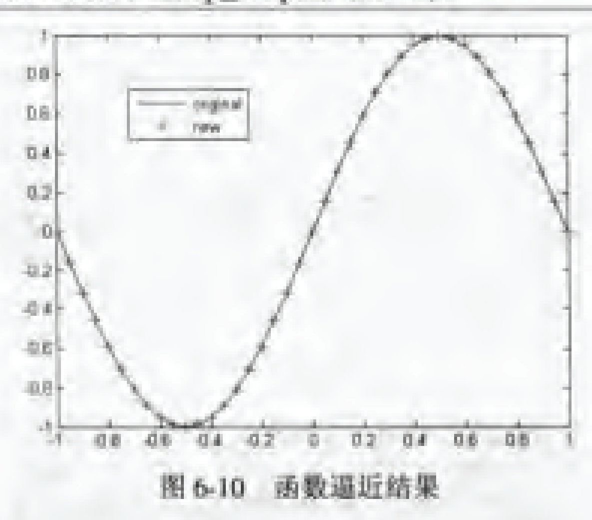


图 6-10 函数逼近结果

6.4 数据操作

数据操作一方面指的是将 MATLAB 工作空间中的数据导入到 GUI 中，也可以将 GUI 中的数据变量导出到 MATLAB 工作空间中；另一方面是指数据的存储、删除、恢复和重新调用等。

6.4.1 工作空间到 GUI 的数据导入

可以通过如下的方式将工作空间中的数据变量导入到 GUI 中。

首先，在 MATLAB 命令行中输入如下的代码：

```
p=-1:0.15:1;
t=sin(pi*p);
```

这样一来，就得到了数据向量 p 和 t，它们都由 41 个元素组成。回到 Network/Data Manager 窗口，单击其中的【Import】按钮，得到如图 6-11 所示的对话框。

选中此对话框左侧的 Source 域中的 Import from MATLAB workplace 单选框，然后在其中的 Select a Variable 文本框中选中 p，在对话框右侧的 Name 文本框中输入 p，表示在 GUI 中该数据向量的名称亦为 p，然后选中 Inputs 单选框，表示该数据向量是网络的输入向量。最后单击右下角的【Import】按钮，就可以将 p 作为网络的输入向量导入到 GUI 中。按照同样的方式，也可以将 t 作为网络的目标向量导入到 GUI 中。此时，新导入的 p 和 t 分别出现在 Network/Data Manager 窗口中的 Inputs 和 Targets 文本框中，分别表示神经网络的输入向量和目标向量。



图6-11 数据输入

此外，该对话框Source域中还提供了另外一个选项Load from disk file，这表明还可以从磁盘中导入数据。



上面的两个数据向量p虽然名称一致，但它们实际上处于不同的工作空间，代表不同的变量。

6.4.2 GUI到工作空间的数据导出

通过GUI得到的网络仿真结果、训练结果和误差等，都可以导出到MATLAB的命令行空间中，在命令行工作空间中可以利用这些数据进行一些其他操作。

以仿真结果的导出为例介绍数据的导出过程。仿真得到数据向量simubp_sim后，回到Network/Data Manager窗口，在右侧的Outputs文本框中出现了simubp_sim变量，如图6-12所示。



图6-12 Network/Data Manager窗口

选中simubp_sim变量，然后单击【View】按钮，可以看到该变量的元素数据，如图6-13所示。



图 6-13 变量显示

单击【Export】按钮，出现数据导出对话框，如图 6-14 所示。这里列出了 GUI 当前所有的数据变量。这里选中变量 simubp_sim，并单击【Export】按钮，将该对话框关闭，变量 simubp_sim 就导出到命令行空间中了。为了检验以上过程是否成功，在 MATLAB 命令行中输入 who 来检查此时工作空间中的所有变量，可得到以下结果：



```
Your variables are:
p     simubp_sim     t
```

这表明变量 simubp_sim 已经成功地导入到命令行空间中了。

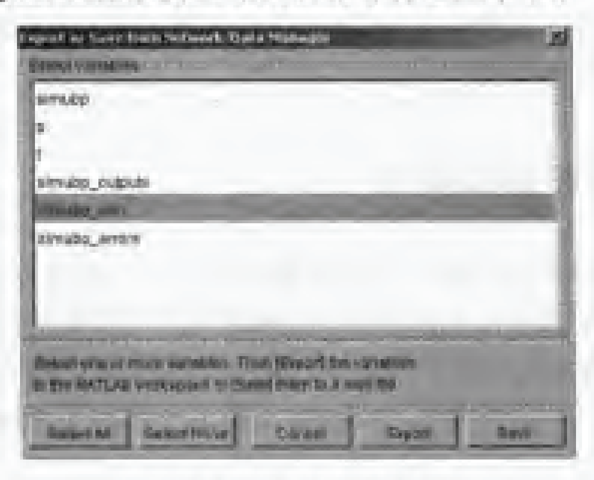


图 6-14 数据导出对话框

可以按照同样的方式将设计好的网络导出到命令行空间中，导出成功后，在命令行窗口中输入 simubp，可得到网络的基本信息，如下：

```
simubp =
    Neural Network object:
    architecture:
    numInputs: 1
    numLayers: 2
    biasConnect: [1; 1]
    inputConnect: [1; 0]
    layerConnect: [0 0; 1 0]
    outputConnect: [0 1]
    targetConnect: [0 1]
    numOutputs: 1   (read-only)
    numTargets: 1   (read-only)
    numInputDelays: 0   (read-only)
    numLayerDelays: 0   (read-only)
......
```

还有很多内容，限于篇幅的原因，在此不再赘述，读者可以自行查看。

6.4.3 数据的存储和读取

通过GUI定义的变量、数据和网络可以存储到硬盘中，需要的时候再加载，这样就可以使得设计好的网络能够重复使用，而不用每次都要重新设计。下面我们以神经网络的存储和读取为例，介绍数据的存储和读取过程。

回到图6-14，选中simubp项将其激活，然后单击右下角的【Save】按钮，出现Save to a MAT file对话框，如图6-15所示。



图6-15 Save to a MAT file对话框

单击【保存】按钮，将网络以*.MAT文件的形式存储在硬盘中，这里将文件名设定为sinsim。

如果不小心将网络误删除或者需要重新启动MATLAB，我们还可以利用导出功能将网络文件导出到GUI中。下面介绍在重新启动MATLAB的情况下，如何将网络文件导出到GUI中。

重新启动MATLAB后，在命令行窗口输入nntool命令，启动Network/Data Manager窗口，单击【Export】按钮，出现如图6-16所示的数据导入对话框。

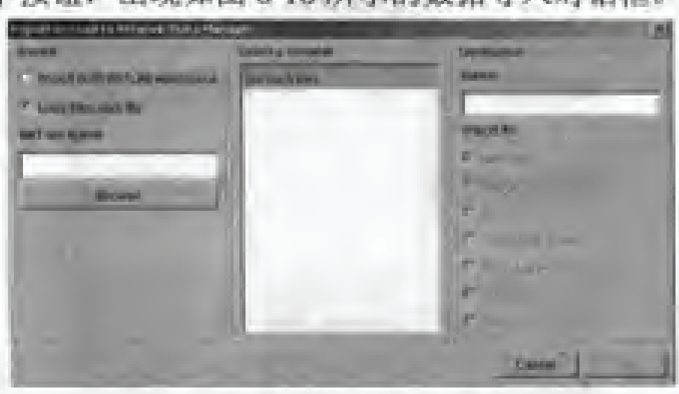


图6-16 数据导入对话框

在Source域中选中Load from disk file单选框，然后单击【Browse】按钮确定网络文件的来源，如图6-17所示，选中sinsim文件，单击【打开】按钮或者双击该文件，将文件导入到GUI中。

单击【打开】按钮后，出现如图6-18所示的窗口，在Select a Variable框中选中simubp，可以在Name文本框中输入网络在GUI中的名称，这里取原来的名称，也就是默认的名称

simubp。MATLAB 有识别数据类型的能力，这里它已经识别出导入的数据类型为神经网络，Import As 下的单选项中只有 Network 是有效的，其余的都是灰色的，不可选。



图 6-17　网络文件选择

最后单击位于窗口右下角的【Load】按钮，我们发现网络 simubp 已经在 GUI 中了，如图 6-19 所示。

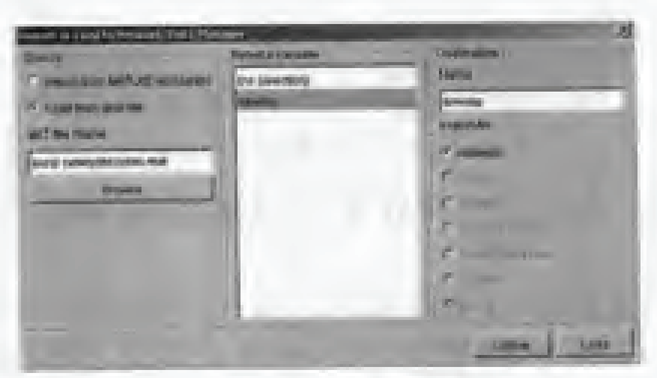


图 6-18　网络文件导入



图 6-19　导入网络 simubp 后的 Network/Data Manager 窗口

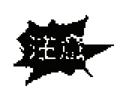
6.4.4　数据删除

在通过 GUI 进行神经网络的设计分析时，有时候可能会出现误操作。例如，在设计网

络输入向量的时候出现了错误，但直到后来才发现，这时候就需要删除错误的输入向量，从而设计新的输入向量。

数据删除最简单的方法是强行关闭 MATLAB 窗口，或者到任务管理器中结束 MATLAB.exe 进程。这种方法的优点是速度非常快，但必须要提醒读者的是，利用这种方法进行数据删除以前，必须将所有认为有用的数据变量保存起来，然后在重新启动 MATLAB 后，将它们一一导入到新的 GUI 中，这样才可以保证数据不丢失。

另一种方法是在 Network/Data Manager 窗口中选中要删除的数据，如图 6-19 所示，选中了网络 simubp，然后单击【Delete】按钮，就可以将 BP 网络 simubp 删除了。



删除 GUI 工作空间中的数据并不影响已经导出到 MATLAB 命令行空间中的数据，即使关闭了 Network/Data Manager 窗口，命令行空间中的数据也依然存在。

6.5 小　　结

GUI 是神经网络工具箱提供的人机交互界面，它引导工程人员一步步地建立和训练网络，避免了代码的编写过程。但是，如果你是一个新手，我建议不要一上来就使用 GUI 来解决实际问题，因为 GUI 中的一些功能只有在熟练掌握了工具箱中的大部分函数后才可以正确应用。因此，最好的方式是首先利用编写代码的方式来学习神经网络工具箱，精通了各种函数的实际意义、调用格式和注意事项以后，就可以利用 GUI 方便快捷地来解决实际问题了。



第 7 章　神经网络控制理论及应用设计

我们已经知道，神经网络由于其固有的自学习、自适应、自组织和大规模并行处理等能力，已经在模式识别、信号处理、系统辨识、控制，以及优化等领域得到了广泛的应用。近年来，神经网络在智能控制系统中显示了极大的应用潜力，人们已经利用神经网络来处理控制系统的非线性、不确定性和如何逼近系统的辨识函数等问题，并取得了大量的研究成果。

根据控制系统的结构，可把神经控制的应用研究分为诸如监督式控制、逆控制、神经自适应控制和预测控制等方向。控制理论领域与神经网络之间的关系如图 7-1 所示，其中的空白框表示没有明显的对应关系。

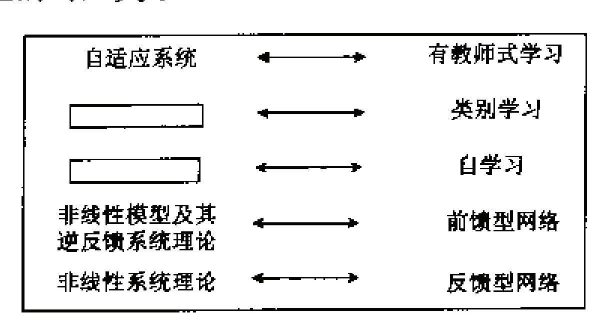


图 7-1　控制与神经网络的关系

本章的内容包括：

- 神经网络控制结构
- 反馈线性化控制及 MATLAB 实现
- 基于 Simulink 的神经网络控制实现

7.1　神经网络控制结构

神经网络用于控制系统设计主要是针对系统的非线性、不确定性和复杂性进行的。由于神经网络的自适应能力、并行处理能力和超强的鲁棒性，使得采用神经网络的控制系统具有更快的计算速度（实时性）、更强的适应能力和更好的鲁棒性。资料显示，国内外将神经网络应用于控制系统设计的方式和结构有很多，目前尚无统一的分类方法。接下来本节将简单介绍一下现有的、应用比较广泛的几种神经网络控制结构。

7.1.1　神经网络监督控制

在实际应用中有许多控制问题，由于被控对象的解析模型位置完全未知或者部分未知，利用传统的控制理论来设计控制器非常困难，人工控制可能是惟一的选择。但是，在恶劣

的工作环境下，或者控制任务只是一些单调、重复、繁重的简单操作时，则有必要利用自动控制取代上述手工操作。

取代人工控制的途径有两种：一种是将手工操作中的经验总结成普通的规则或模糊规则，然后构造相应的专控控制或模糊控制器；另一种是在知识难以表达的情况下，利用神经网络学习人的控制行为，即对人工控制器建模，然后利用此神经网络控制器代替人工控制。这种通过对人工或传统控制器进行学习，然后利用神经网络控制器取代或逐渐取代原控制器的方法，称为神经网络监督控制。这类方案的示意图如图 7-2 所示。

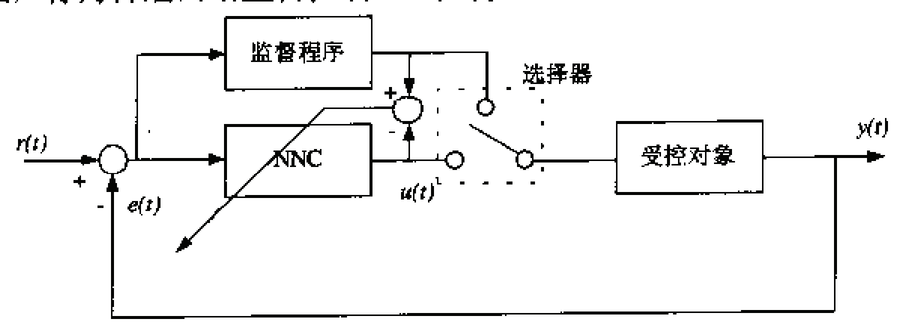


图 7-2 神经网络监督控制（I）

可见，神经网络监督控制实际上就是建立人工控制器的正向模型。经过训练，神经网络记忆该控制器的动态特性，并且接收传感信息输入，最后输出与人工控制器相似的控制作用。此结构的缺点在于人工控制器靠视觉反馈进行控制，而利用神经网络控制器进行控制后，由于缺乏视觉反馈，所构成的控制系统实际上是一个开环系统，这使得它的稳定性和鲁棒性均得不到保证。

另一种神经网络监督控制是传统控制器，如在 PID 控制器的基础上，增加一个神经网络控制器，如图 7-3 所示。此时，神经网络实际上是一个前馈控制器，它建立的是被控对象的逆模型。可见，神经网络控制器通过向传统控制器的输出进行学习，在线调整自己，目标是使反馈误差 $e(t)$或 $u(t)$趋向于 0，从而使自己逐渐在控制中占据主导地位，以便最终取消反馈控制器的作用。但与图 7-2 的结构不同，这里仍然存在反馈控制器，一旦系统出现干扰等情况，反馈控制器依然可以发挥作用。因此，采用这种前馈加反馈的监督控制方法，不仅可以确保控制系统的稳定性和鲁棒性，而且可以有效地提高系统的精度和自适应能力。

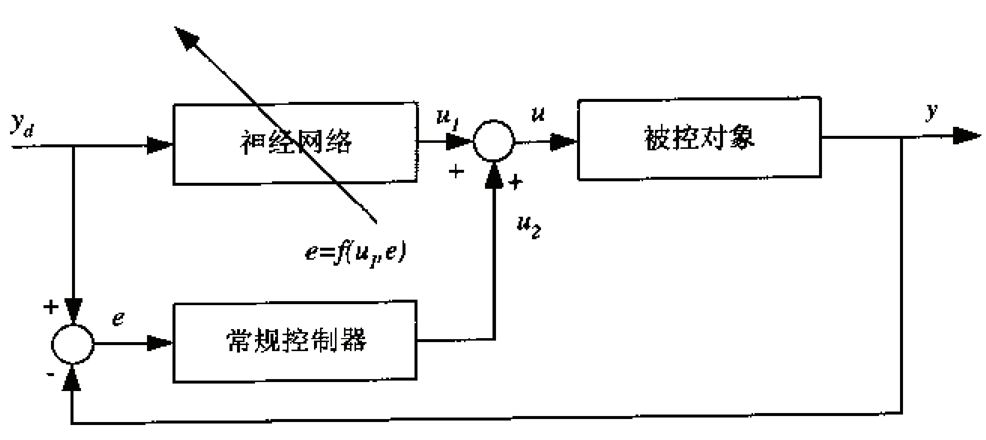


图 7-3 神经网络监督控制（II）



7.1.2 神经网络直接逆控制

神经网络直接逆控制就是将被控制对象的神经网络逆模型直接与被控对象串联起来，以使得期望输出（即网络输入）与对象实际输出之间的传递函数等于 1。直接逆控制在机器人控制方面有着较多的应用。

神经网络直接逆控制在结构上与逆向建模思想有很多相似之处。显然，该方法的可用性在相当程度上依赖于逆模型的准确程度。由于缺乏反馈，简单连接的直接逆控制将缺乏鲁棒性。因此，一般应使其具有在线学习的能力，即必须保证逆模型的连接权值能够在线修正。

如图 7-4 所示，它给出了两种结构方案。在图 7-4（a）中，NN1 和 NN2 为两个网络结构完全相同的神经网络，采用相同的学习算法，以使得 NN2 的输出和输入 u 的偏差 e 的二次型达到最小，即 NN1 和 NN2 都沿着 $E=\frac{1}{2}\sum_{t}e^{\mathrm{T}}(t)e(t)$ 的负梯度方向修改连接权值。在图 7-4（b）中，评价函数一般采用 $e(t)=y_d(t)-y(t)$，但也可采用 $e(t)=M_y(y_d(t)-y(t))+M_u u(t)$等更为一般的权值评价函数，其中 M_y 和 M_u 为适当维数的矩阵。

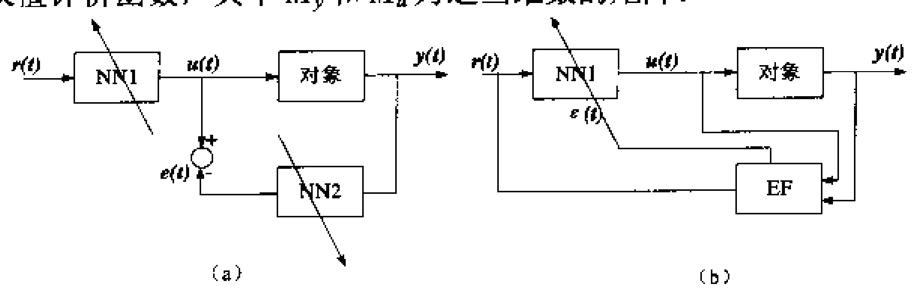


图 7-4 NN 直接逆控制

7.1.3 NN 自适应控制

与传统自适应控制相同，神经网络自适应控制也可分为自校正控制（STC）和模型参考自适应控制（MRAC）两种。两者的区别在于，自校正控制将根据对系统正向和（或）逆模型建模的结果，直接调节控制器的内部参数，使系统满足给定的性能指标。而在模型参考控制中，闭环控制系统的期望性能由一个稳定的参考模型描述，控制的目的是使被控对象的输出渐进地趋于参考模型的输出。

1. NN 自校正控制

神经网络自校正控制可分为直接和间接控制两种。它们的根本区别是：前者使用常规控制器，离线辨识的神经网络估计器需要足够高的建模精度；而后者则同时使用神经网络控制器和神经网络估计器，并且估计器可以在线修正。

神经网络直接自校正控制也称为神经网络直接逆控制，实际上它们是一致的。其结构如图 7-4 所示。

神经网络间接自校正控制结构如图 7-5 所示。神经网络估计器用做过程参数或某些非线性函数的在线估计器，而控制信号则通过常规控制器发出。这种控制结构已被许多研究

者利用不同的网络模型，针对不同的控制对象而使用。

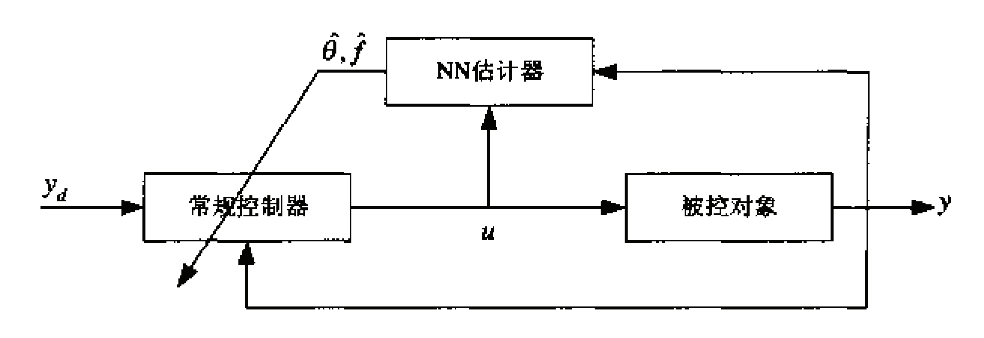


图 7-5　神经网络间接自校正控制结构

2. 神经网络模型参考自适应控制

基于神经网络的模型参考自适应控制也可分为直接和间接两种，如图 7-6 所示。神经网络控制器 NNC 的权值修正目标是使输出误差$e_c(t)=y(t)-y^m(t)\to 0$，或者使 e_c 的二次型最小。对于直接模型参考自适应控制，如图 7-6（a）所示，误差 e_c 的反向传播必须确知被控对象的数学模型，这给 NNC 的训练带来了困难。为解决这一问题，可采用正逆建模的有关方法，引入神经网络辨识器 NNI，首先，为离线辨识被控对象建立正向模型，作为 NNI，可由 e_c 进行在线修正，从而形成如图 7-6（b）所示的间接模型参考自适应控制。显然，在这种结构中，NNI 可为 NNC 提供误差 e_c 或其梯度的反向传播通道。由于参考模型可视为期望输出，因此，在对象部分已知的情况下，若用常规控制器代替 NNC，此法也就与间接自校正控制雷同了。

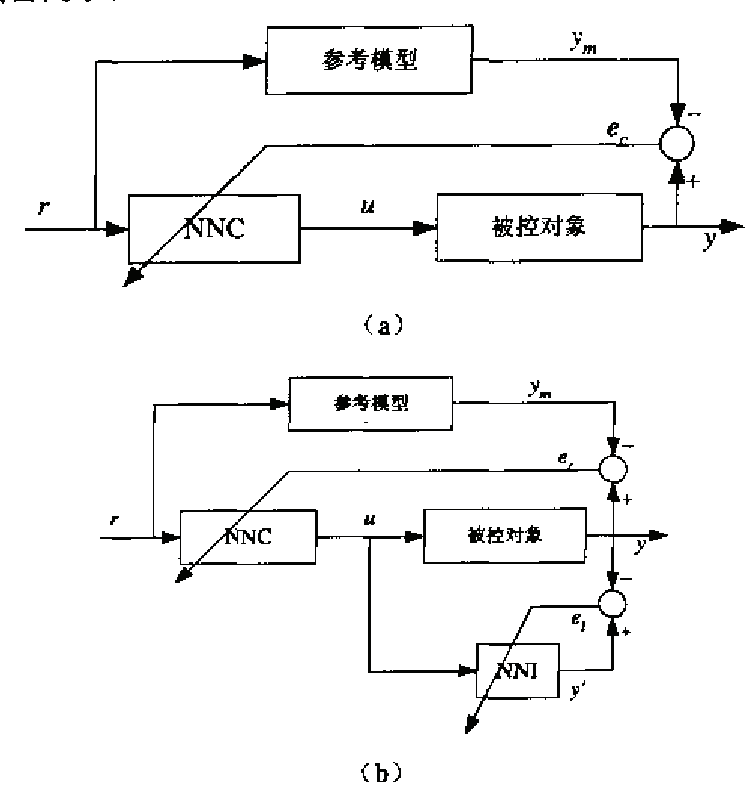


图 7-6　神经模型参考自适应控制



7.1.4 神经网络内模控制

内模控制是近年来被人们逐渐熟悉的一种过程控制方法，它主要利用受控对象模型的逆来构成控制系统。内模控制为非线性反馈控制器的设计提供了一种直接方法，具有较强的鲁棒性。利用神经网络建立受控对象的正向模型和控制器（逆模型），即构成了神经网络内模控制，如图 7-7 所示。

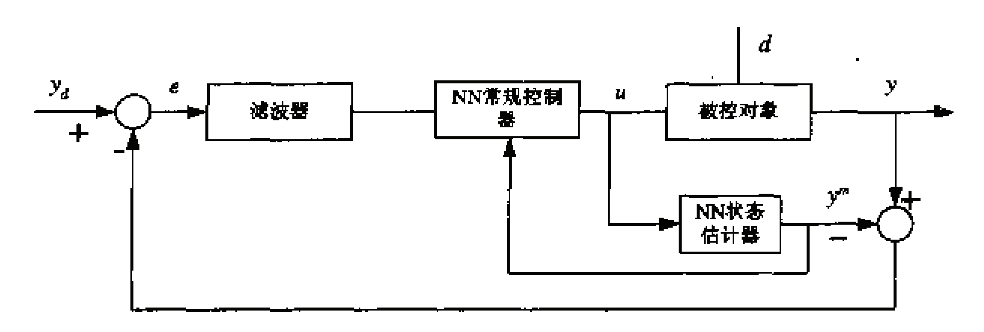


图 7-7　NN 内模控制

在图 7-7 中，系统的正向模型与受控对象并联，两者之差用于反馈信号，此反馈信号又由前馈通道及控制器处理，引入滤波器的目的是为了获得更好的鲁棒性和跟踪响应。

7.1.5 神经网络预测控制

预测控制，又称为基于模型的控制，是 20 世纪 70 年代后期发展起来的一类新型控制算法。这种算法的本质特征是预测模型、滚动优化和反馈校正。已经证明，这种方法对非线性系统有稳定作用。

如图 7-8 所示，给出了神经网络预测控制方案，图中的神经网络预测器建立了被控对象的预测模型，并且可以在线修正。利用预报模型，根据系统当前的输入输出信息，预测未来的输出值。利用神经网络预测器给出的未来一段时间内的输出值和期望输出，对定义的二次型指标函数进行滚动优化（由优化器完成，实质上是一种优化算法，也可利用动态网络实现），产生对未来的控制序列，并仅以第一个控制量进行下一步的控制。

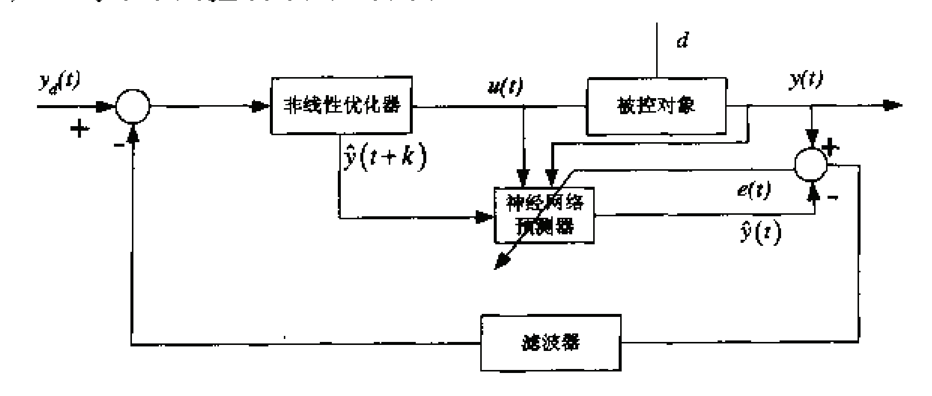


图 7-8　NN 预测控制结构

7.1.6 神经网络自适应评判控制

上述控制方法都有一个本质上的共同点，即都要求提供对象的期望输出。

由神经网络理论可知，神经网络学习算法一般可分为两种类型：有教师学习和无教师学习。有教师学习虽然有最高的学习效率，但它需要教师提供网络的期望输出，对于神经网络控制器，也就是需要提供期望的控制信号。而一般来说，控制的目标就是要找到被控系统的这一期望输入。在系统模型未知或部分未知的情况下，这显然是难以预先提供的。无教师学习实质上是一种自组织聚类方法，它利用输入数据构造内部教师模型，不接受其他信息。除了这种学习方式外，还存在一种再励学习（reinforcement learning）方式，实质上也可认为是一种有教师学习。两者的区别在于，有教师学习中教师需要对给定输入提供期望输出；而再励学习中的教师不是指定的，而是一种评价，即教师是根据一定的性能指标对系统性能做出评价，提供给网络进行学习。

显然，只需要一个评价值的再励学习，在缺乏被控系统精确观测值并只能获得定型的信息反馈时，是十分有用的。

神经网络自适应评判控制就是利用再励学习的一种控制结构，如图 7-9 所示。这种方法最早由 Barto 等在 1983 年提出，后由 Anderson 等加以发展。从图中可以看出，神经网络自适应评判控制有两个组成部分，其中自适应评判网络在整个控制系统中，相当于一个需要进行再励学习的教师。它的作用一方面是通过不断奖励、惩罚等再励学习，使自己逐渐成为一个合格的教师；另一方面是在学习完成后根据被控系统的当前状态及外部再励信号，产生一个内部再励信号，对目前的控制作用做出评价，并据此知道控制作用网络的再励学习，产生下一步的控制作用。

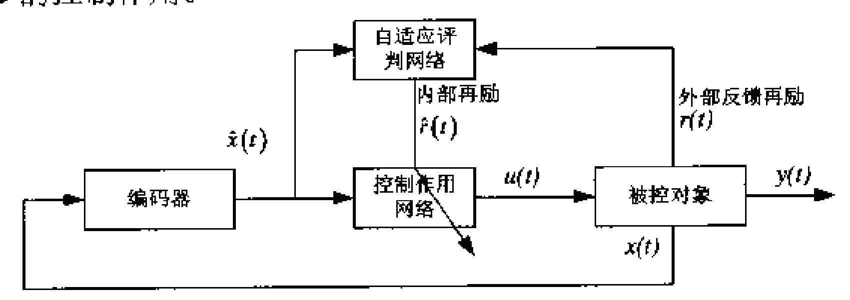


图 7-9 神经网络自适应评判控制

神经网络自适应评判控制与人脑的控制与决策过程比较接近，除了应该随时了解一些定性信息外，它完全不需要被控对象的先验模型，特别适合于许多具有高度非线性和严重不确定性的复杂系统的控制。

除了上面介绍的几种神经网络控制结构，实际应用中还存在许多其他类型的结构，例如将神经网络放置到传统控制器之前或之后的串联结构，将神经网络应用于变结构控制系统设计而形成的神经网络变结构控制等。此外，神经网络与专家系统、模糊控制、遗传算法等技术相结合形成的神经网络专家控制、神经网络模糊控制、基因神经网络模糊控制等，近年来已经受到广泛的重视并成为智能控制技术研究和发展的新热点。

实际上，就目前来说，神经网络控制尚无统一的分类方法，Werbos 将其分为学习控制、直接逆动态控制、神经自适应控制、BTT 控制和自适应评判控制 5 类。另外还有一些分类



方法，不过都是基于控制结构形式上进行划分的。如果分类时考虑设计的出发点，那么可将神经网络控制分为以下3类。

（1）单纯的神经网络控制

它的特点是不依赖传统控制理论和控制系统结构，完全从神经网络自身特点出发，构成控制系统。神经网络监督控制、神经网络直接逆控制（自校正控制）和神经网络自适应评判控制均属于这一类。

（2）基于传统控制理论的神经网络控制

它的特点是采用传统控制理论的控制结构，用神经网络取代其中的部分内容，如辨识器、控制器或对象模型。它包括神经网络间接自校正控制、神经网络模型参考自适应控制、神经网络内模控制、神经网络预测控制、神经网络变结构控制和推理控制。

（3）与其他智能技术相结合的神经网络控制

它的特点是将神经网络与其他智能技术相结合，形成具有各种技术优点的综合智能控制系统，这代表了智能控制的发展方向。常见的有神经网络专家系统、神经网络模糊控制等。

7.2 反馈线性化控制及 MATLAB 实现

反馈线性化控制器又称为 NARMA-L2 控制器。反馈线性化理论的中心思想是通过消除非线性，将一个非线性系统转换为线性系统。

7.2.1 基于神经网络的反馈线性化控制原理

所谓反馈线性，顾名思义就是利用反馈的控制手段来消除系统中的非线性，以使得其闭环系统的动力学方程是线性的。反馈线性化的方法可以很容易地被实施到可控标准型的控制系统中。给定一个系统，其动力学方程为：

$$\frac{d^n x}{dt^n}=f(X)+b(X)u$$

其中，u 是控制输出，x 为标量，X 为状态向量，有：

$$X=\left[x,\frac{dx}{dt},\cdots,\frac{d^{n-1}x}{dt^{n-1}}\right]$$

由此，可将动力学方程写为：

$$\frac{d}{dt}\begin{bmatrix}x_1\\x_2\\\cdots\\x_{n-1}\\x_n\end{bmatrix}=\begin{bmatrix}x_2\\x_3\\\cdots\\x_n\\f(X)+b(X)u\end{bmatrix}$$

此时，希望通过加入一个反馈控制项 u，使得该系统具有以下理想的线性特性：

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \cdots \\ x_n \\ -k_1x_1 - k_2x_2 - \dots - k_nx_n + Cr \end{bmatrix}$$

其中，k_i（i=1,2,...,n）和 C 均为常数，r 为参考输入。

通过两式之间的对比即可求得反馈控制项 u，有：

$$u = \frac{\left[-K^{\mathrm{T}}X + Cr - f(X)\right]}{b(X)}$$

利用 u 对系统进行控制，原系统中的非线性将被消除，被控系统的闭环特性将呈现出其网络的线性特性，并通过选择适当的参数 K 和 C，即可得到期望的任意 n 阶线性系统的响应特性。

7.2.2 反馈线性化控制实例

本小节以神经网络工具箱中附带的简单机械臂为例，介绍基于神经网络的反馈线性化控制的过程。

1. 背景概述

如图 7-10 所示，描述了一个简单的单连接机械臂，控制目标是使机械臂能够自由运动。

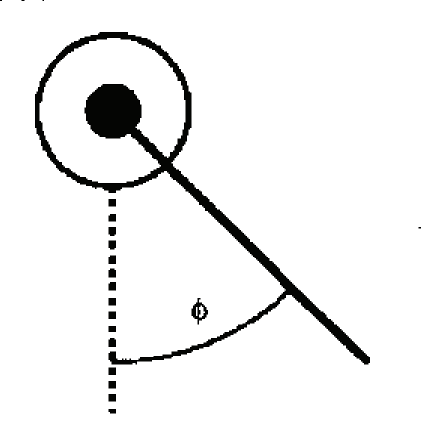


图 7-10　简单的单连接机械臂

该系统的动力学方程为：

$$\frac{\mathrm{d}^2\Phi}{\mathrm{d}t^2} = -10\sin\Phi - 2\frac{\mathrm{d}\Phi}{\mathrm{dt}} + u$$

其中，Φ 代表机械臂的角度，u 代表 DC（直流）电机的转矩。控制目标是训练控制器，使得机械臂能够跟踪如下的参考模型：

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -9x_1 - 6x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 9r \end{bmatrix}$$

其中，r 代表期望的输出角位移。由此可得到：

$$u = 9r - [9 \quad 6]X - f(X)$$

由被控系统的数学模型，可以根据反馈线性化的思想推导出线性化控制系统的控制律，首先取：

$$f(X) = 10\sin x_1 - 2x_2$$

接下来通过设计一个神经网络来进行该反馈线性化控制。

2. 神经网络设计

假定非线性系统的采样周期为 0.05 秒，通过利用：

$$\frac{dX}{dt} = \frac{X(k+1) - X(k)}{\Delta t}$$

的近似关系将被控系统的离散时间动力学特性近似描述为：

$$X(k+1) = X(k) + \Delta t \begin{bmatrix} x_2(k) \\ 10\sin x_1(k) - 2x_2(k) + u(k) \end{bmatrix}$$

或者

$$X(k+1) = F(X(k)) + G(X(k))u(k)$$

接下来设计一个神经网络代替函数 F，利用该函数作为控制器 u 中的一部分抵消系统中的非线性，该函数为（称为逼近函数）：

$$F(X) = x_2 + \Delta t(10\sin x_1 - 2x_2)$$

假定神经网络对该函数的逼近为 N_f，那么带有神经网络的非线性消除控制律为：

$$u = \frac{x_2 + \Delta t(9r - [9 \quad 6]X) - N_f}{\Delta t}$$

显而易见，所需的神经网络具有两个输入，分别为 x_1 和 x_2。网络设计的目的是产生逼近函数的非线性输出，所以输出层只有一个输出。因此，这里设计一个单隐层的 BP 神经网络模型，中间层取 9 个神经元，传递函数为 tansig；输出层神经元的传递函数为 purelin。

为了对网络进行训练，必须设置网络的训练样本。由于非线性函数表达式已经明确给定，所以可以通过随机选取一定数目的工作区的输入，然后用 $F(X)$求得目标输出即可。在实际应用中，如果出现很难描述非线性特性的情况，可通过记录实际被测系统的输入/输出来获得样本数据，再采用适当的控制策略来解决问题。

MATLAB 代码为：

```
x1=rands(400)*pi;
x2=rands(300)*pi;
x3=zeros(100,1)*pi;
p1=x1;
p2=[x2;x3];
P=[p1 p2];
deltatt=0.05;
T=p2+deltatt*(10*sin(p1)-2*p2);
Q=P';
net=newff(minmax(Q),[9 1],{'tansig','purelin'});
net.trainParam.epochs=2000;
net=train(net,Q,T');
```

训练样本的容量为 400 组，其中 300 个是由 0 到 - π之间的 x_1 以及 - π 到π之间的 x_2 所产生；另外 100 个是令速度 x_2 为 0，即稳态时，通过同样范围的 x_1 产生，也就是说，希望网络在系统稳态时的非线性输出要更加精确，以便将其通过反馈控制而完全抵消，使其系统输出达到零稳态误差。

网络的训练误差曲线如图 7-11 所示。由图可见，经过 1000 次训练后，网络的输出误差曲线的收敛速度已经比较缓慢了，此时的误差也比较小了。

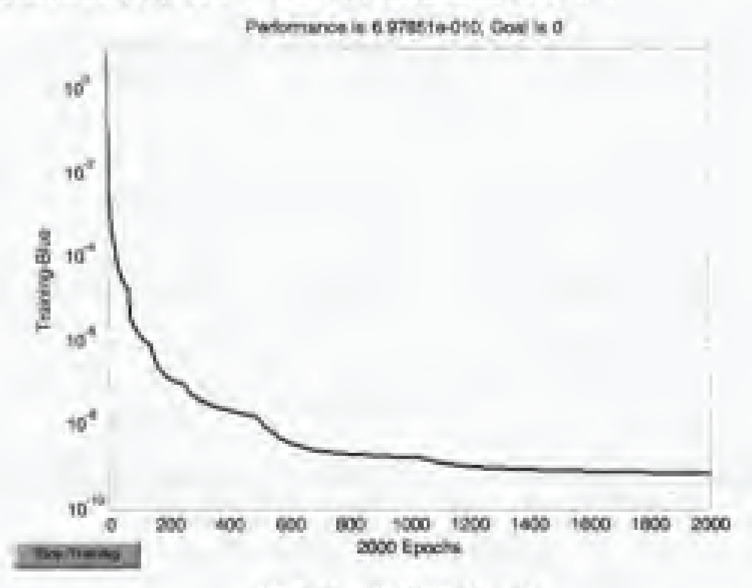


图 7-11　训练误差曲线

为了检验网络对函数 F 的逼近能力，分别在 x_2=0 和 x_1= π /2 时对函数进行逼近，x_2=0 表示 N_f只作为 x_1 的函数，x_1= π /2 表示将位置固定在水平方向，N_f是速度 x_2 的函数。逼近情况分别如图 7-12 和图 7-13 所示。图中的*号表示函数 F 的输出，线条表示神经网络的输出。由图可见，网络能够很好地逼近非线性函数。

获得了性能良好的神经网络后，接下来就是利用该神经网络的输出求解该控制系统的响应特性，然后与其他的控制方式进行比较。不难发现，带有神经网络的控制器的鲁棒性更高，精确性也更好。这已经超出本书的范围，在此就不详细说明了。

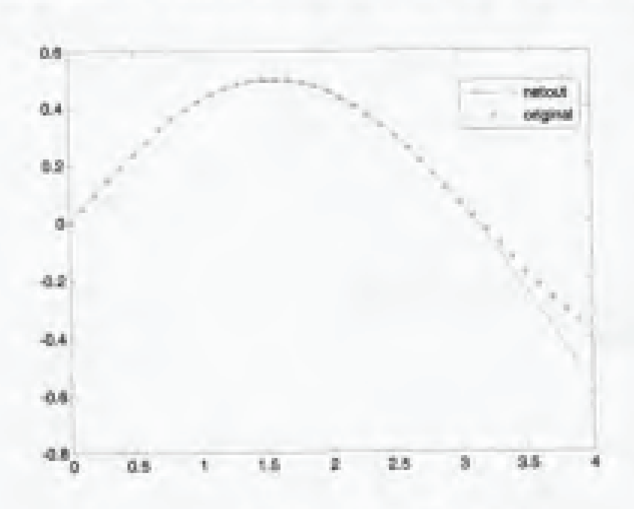


图 7-12 x_2=0 时网络的逼近性能

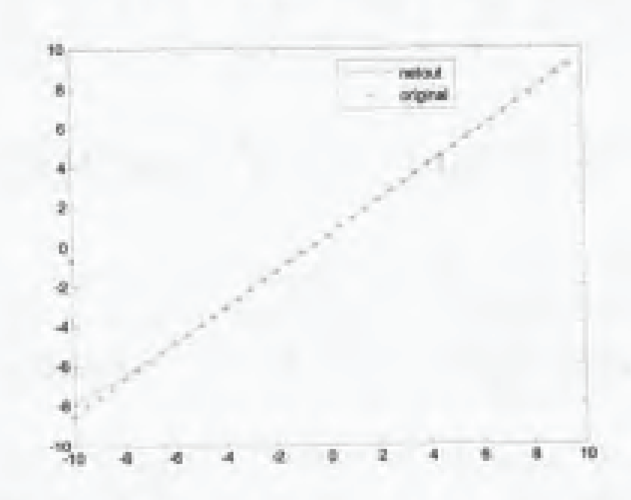


图 7-13 $x_1=\pi/2$ 时网络的逼近性能

本实例的 MATLAB 代码为：

```
x1=rands(400)*pi;
x2=rands(300)*pi;
x3=zeros(100,1)*pi;
p1=x1;
p2=[x2;x3];
P=[p1 p2]';
deltatt=0.05;
T'=p2+deltatt*(10*sin(p1)-2*p2);
net=newff(minmax(P),[9 1],{'tansig','purelin'});
net.trainParam.epochs=2000;
net=train(net,P,T);
%利用 x2=0 的测试样本求网络的仿真输出
a=0:0.1:3.9;
b=zeros(1,40);
P_test=[a;b];
```

```
y=sim(net,P_test);
nf=0.5*sin(a);
plot(a,y);
hold on
plot(a,nf,'*');
hold off
figure;
%利用 x1=π/2 的测试样本求网络的仿真输出
a=ones(1,40)*0.5*pi;
b=-10:0.5:9.5;
P_test=[a;b];
y=sim(net,P_test);
nf=0.5+0.9*b;
plot(b,y);
hold on
plot(b,nf,'*');
hold off
```

7.3 基于 Simulink 的神经网络控制

神经网络工具箱提供了一组可以在 Simulink 中使用的模块，它们可以用来建立网络和基于网络的控制器。工具箱中还提供了 3 类控制的有关 Simulink 实例，分别为模型预测控制、模型参考控制和反馈线性化控制，本节以基于神经网络的模型参考控制为例，介绍如何利用 Simulink 实现神经网络控制。

模型预测控制 MPC 算法是基于被控对象的将来状态或输出的动态预测值，在线求解当前控制量的一种自适应控制方法。目前，针对线性系统模型的预测问题已经得到了解决。但是对于非线性系统的模型预测控制，由于需要建立非线性系统的预测模型，需要解决的困难还很多。神经网络作为一种新型的方法体系，具有分布并行处理、非线性映射、自适应学习和鲁棒容错等优良特性，为解决非线性系统的模型预测控制问题提供了一种新的可能的途径。

7.3.1 基于神经网络的 MPC 原理

MPC 是 20 世纪 70 年代后期发展起来的一类新型的计算机控制算法，这种算法的本质特征包括 3 个要素：预测模型、滚动优化和反馈校正。

1. 预测模型

在模型预测控制中需要一个描述系统动态行为的基础模型，称为预测模型。这个预测模型应该具有根据系统的历史信息和未来输入，预测其未来输出值的功能。

2. 滚动优化

预测控制实质上是一种优化控制算法，它采用滚动式的有限时域的输出优化。这种优

化方式有两个特点。其一，优化目标是随时间推移的。也就是说，预测控制在每个时刻都提出一个基于该时刻的优化指标，而不是采用全局优化指标。滚动优化指标的局部性，虽然使其在理想情况下只能得到全局的次优解，但是当模型失配或者存在时变、非线性或干扰因素时，能够顾及这种不确定性，及时进行弥补，减小偏差，保持实际上的最优化控制。其二，由于采用了有限时域的输出优化，结合模型的输入/输出功能，易于得到简便的在线控制律，适合在线反复进行优化的需要。

3. 反馈校正

预测控制在控制的每一步，都需要检测实际输出并与基于预测模型的预测值相比较，以此修正模型预测的不确定性，然后进行新的优化。因为作为基准的预测模型只是系统动态特性的粗略描述，加上实际系统中可能存在的非线性、时变和干扰等因素，基于模型的预测不可能和实际完全相符，这就需要基于偏差的误差预测对模型预测进行补充，使得优化建立在较准确的预测基础上。这种双重预测手段是克服系统中存在的不确定性的一种有效手段。

由于在模型预测控制中需要一个描述系统动态行为的预测模型，而神经网络可以对非线性动态过程进行精确描述，因此，利用神经网络对非线性系统进行系统辨识，可以较好地解决非线性系统的预测模型问题。构建神经网络预测模型是设计神经网络预测控制器的第一步，用做预测模型的神经网络结构可以为带有输入延迟的网络结构，如图 7-14 所示。

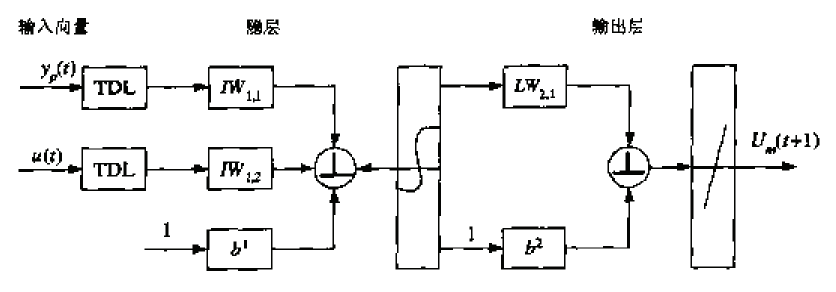


图 7-14　神经网络结构

经过对神经网络的训练，通过它可以辨识出控制对象对各种输入信号的动态响应特性。利用神经网络进行系统辨识的过程如图 7-15 所示。其中，u 为控制信号，y_p 为期望响应，y_m 为神经网络响应。神经网络预测模型应该具有根据系统的当前控制输入和输出信息来预测未来输出值的功能。预测模型建立后，接下来就是构建模型预测控制器，模型预测控制过程如图 7-16 所示。

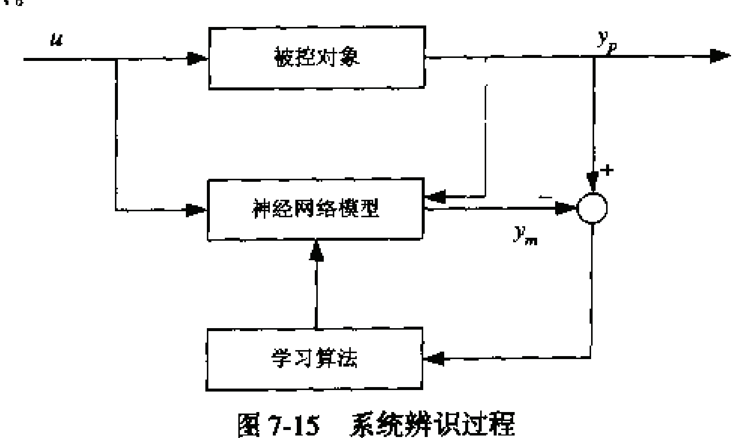


图 7-15　系统辨识过程

综上所述，基于神经网络的预测控制系统设计的主要过程可以分为如下几步。

（1）构建被控对象的系统模型。

（2）根据被控对象的系统模型构建神经网络预测模型，即神经网络系统辨识过程。

（3）设置非线性优化器。

（4）建立反馈控制系统模型，对系统进行仿真，根据仿真结果，调整非线性优化器的各项参数设置。

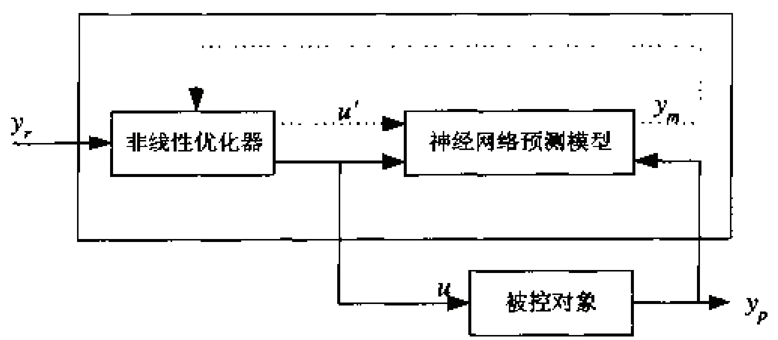


图 7-16　模型预测控制过程

7.3.2　模型预测控制实例

神经网络工具箱中给出了一个模型预测控制的实例，该实例的背景为水反应器，它是一个非线性系统。神经网络工具箱结合了 Simulink 模块，实现了该系统基于神经网络的模型预测控制，控制器可以计算控制输入，然后预测系统在未来某个时间段中的性能。

1. 背景概述

这里研究的背景系统是一个水反应器，如图 7-17 所示。该系统的动力学模型为：

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = w_1(t) + w_2(t) - 0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t} = (C_{b1} - C_b(t))\frac{w_1(t)}{h(t)} + (C_{b2} - C_b(t)\frac{w_2(t)}{h(t)} - \frac{k_1 C_b(t)}{(1 + k_2 C_b(t))^2}$$

其中 $h(t)$ 为液面高度，$C_b(t)$ 为产品输出浓度，$w_1(t)$ 为浓缩液 C_{b1} 的输入流速，$w_2(t)$ 为稀释液 C_{b2} 的输入流速。输入浓度设定为：$C_{b1} = 24.9$，$C_{b2} = 0.1$。消耗常量设置为 $k_1 = 1$，$k_2 = 1$。

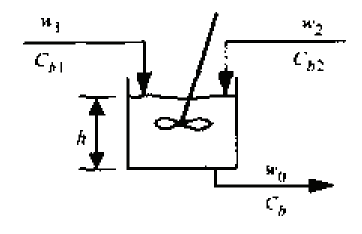


图 7-17　水反应器

控制的目标是通过调节流速 $w_2(t)$ 来保持产品浓度。为了简化演示过程，不妨设 $w_1(t) = 0.1$。出于简化的原因，在本例中不考虑控制液面高度 $h(t)$。

2. 模型创建

在MATLAB命令行窗口中输入predcstr后按回车键，或者到MATLAB的Demos标签页中，在Neural Network下找到Control Systems子节点，然后单击Predctive Control of a tank reactor(sim)的链接，出现如图7-18所示的窗口，单击Open this model链接，出现如图7-19所示的控制模型窗口。

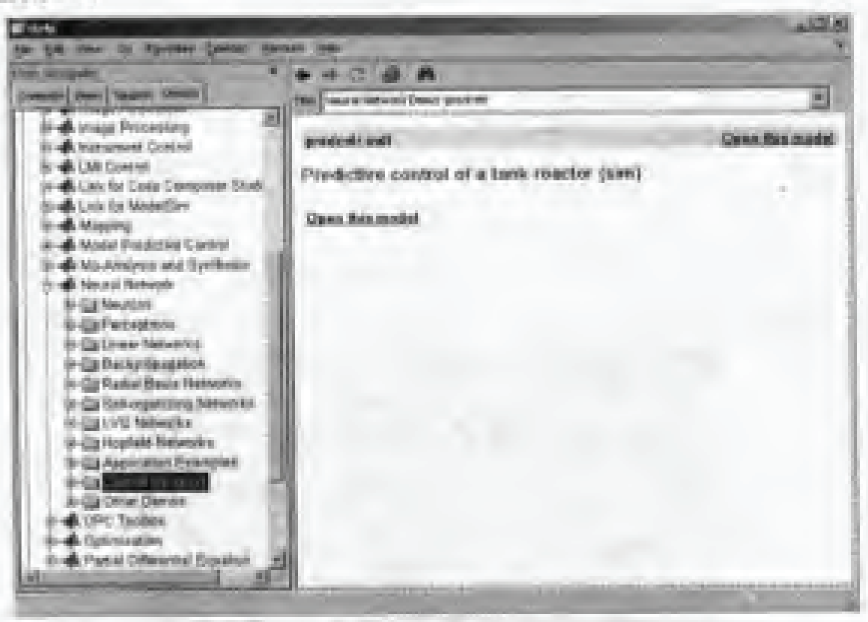


图7-18 启动窗口

神经网络控制模型已经在图7-19所示的窗口中了。图中的深色部分表示神经网络预测控制器，Random Reference用于产生随机输入信号，Plant就是被控对象，这里为水反应器。双击该对象，所出现窗口中描述的内容为该系统的Simulink模型，如图7-20所示。由于Simulink的原理不属于本书的内容，所以这里不做详细介绍，感兴趣的读者可参考其他资料或者查阅MATLAB的帮助文档。

神经网络预测控制器模块是在神经网络工具箱中生成并复制过来的，该模块的Control Signal结点和系统模型的输入结点相连接，Plant Output和系统模型的输出结点相连接，Reference结点连接了信号发生器Random Reference的输出端。

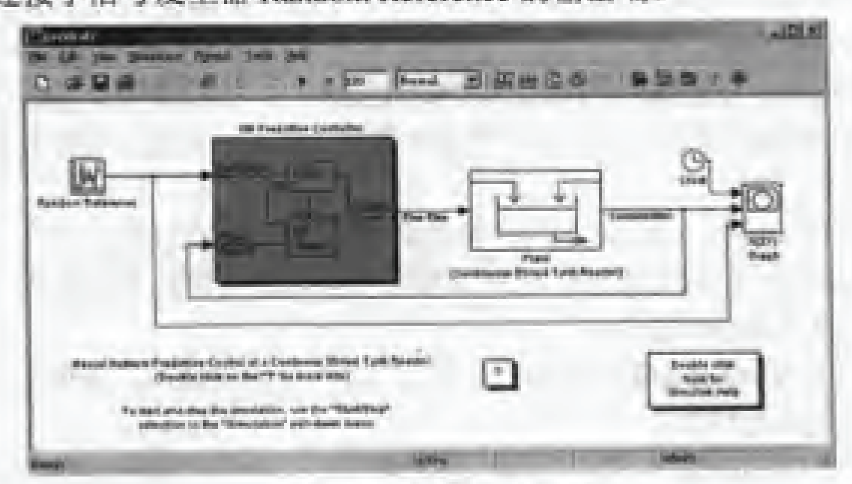


图7-19 控制模型窗口

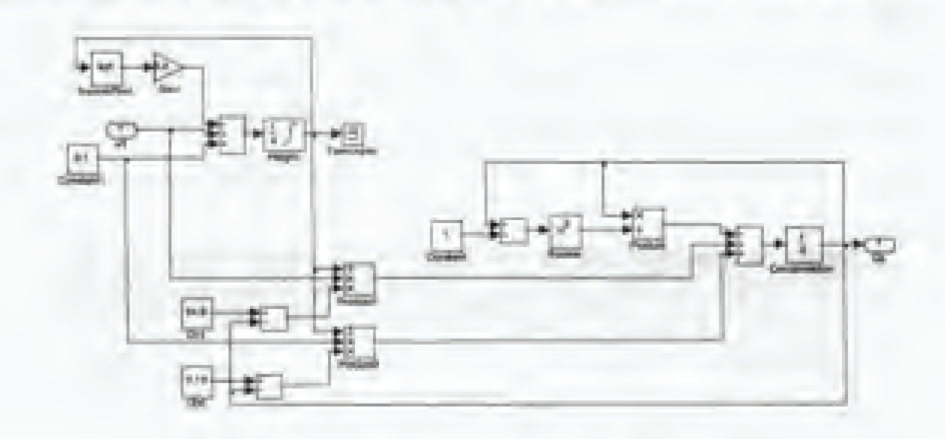


图 7-20　水反应器的 Simulink 模型

双击该控制器模块，出现如图 7-21 所示的参数设置窗口。该窗口用于设置控制器的参数。

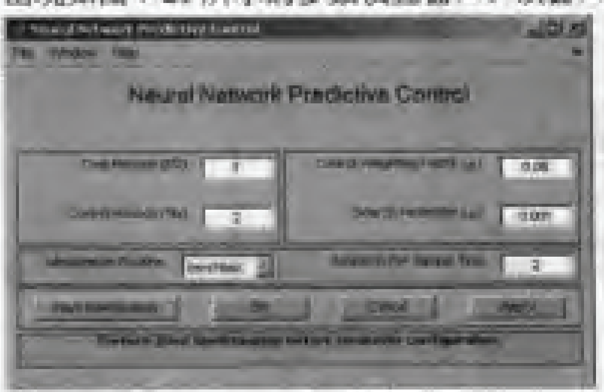


图 7-21　控制器的参数设置窗口

在本窗口中，可以调整的参数有：N2、Nu、α 和权重参数 ρ，其中 α 用于控制最优性，它决定了一个成功的优化步骤，在性能上需要多大的阻尼。此外，在该窗口中，可选择用于优化计算的线性最小化算法，还可设定在每个采样时间中进行多少次迭代优化算法。

该窗口带有解释功能，当把鼠标移到某个参数的名称上时，就会出现对于该参数的解释，归纳如下：

- Cost Honizon（N2）：指定时间步数，在此期间内预测误差达到最小；
- Control Honizon（Nu）：指定时间步数，在此期间内控制增量达到最小；
- Minimization Routine：可以从几个线性搜索程序中选择一个用做最优化算法；
- Control Weight Factor（ρ）：在性能函数中，控制权重因子用于与控制增量的平方和相乘；
- Search Parameter（α）：这个参数决定了线性搜索何时停止；
- Iterations Per Sample Time：选择在每个采样时间中优化算法迭代的次数。

下面是对于 4 个按钮的说明：

- 【Plant Identification】：单击此按钮可打开系统辨识窗口，在使用控制器之前，系统必须先进行辨识；
- 【OK】、【Apply】：在设定好控制器参数以后，单击这两个按钮的任一个都可以将这些参数导入 Simulink 模型；

- 【Cancel】：取消刚才的设置。

3. 系统辨识

单击对话框中的【Plant Identification】按钮，出现系统辨识参数设置窗口，如图 7-22 所示。实际上这已经调用了 MATLAB 的系统辨识工具箱。

由图 7-22 可见，整个窗口分为 3 个部分，最上面的为 Network Architecture，用于确定网络的结构；中间的为 Training Data，用于建立训练样本数据；最下面的为 Training Parameters，用来设定训练参数。

下面简单介绍窗口参数及按钮的含义。其参数含义归纳如下：

- Size of Hidden Layer：设置在系统模型网络第一层中的神经元数目；
- Sampling Interval（sec）：指定程序从 Simulink 模型中采集数据的间隔；
- Normalize Training Data：指定是否使用 premnmx 函数来将数据标准化；
- No.Delayed Plant Inputs：指定加到系统网络模型的输入延迟；
- No.Delayed Plant Outputs：指定加到系统网络模型的输出延迟；
- Training Samples：指定为训练而产生的数据点的数目；
- Maximum Plant Input：指定随机输入的最大值；
- Minimum Plant Input：指定随机输入的最小值；
- Maximum Interval Value（sec）：指定一个最大的间隔，在这个间隔中，随机输入将保持不变；
- Minimum Interval Value（sec）：指定一个最小的间隔，在这个间隔中，随机输入将保持不变；
- Limit Output Data：用于选择系统输出是否为有界值；
- Maximum Plant Output：指定输出的最大值；
- Minimum Plant Output：指定输出的最小值；
- Simulink Plant Model：指定用于产生训练数据的模型（.mdl 文件）；
- Training Epochs：指定训练迭代的次数；
- Training Function：指定训练函数；

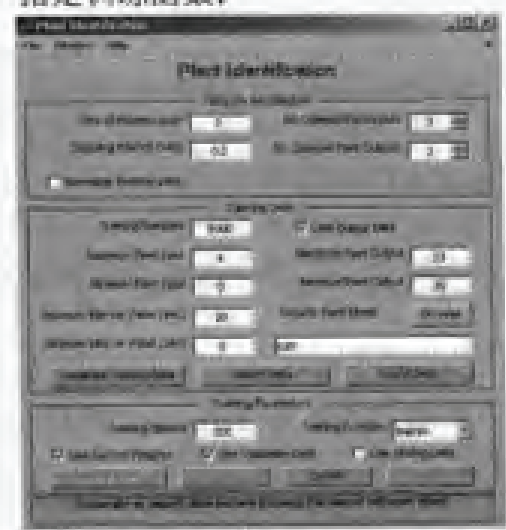


图 7-22 系统辨识参数设置窗口

- Use Current Weights：指定是否选择当前的权重用于连续训练；
- Use Valid Data：指定是否选择合法数据停止训练；
- Use Testing Data：指定在训练过程中测试数据是否被追踪。

下面是关于按钮的说明：

- 【Generate Training Data】：产生用于网络训练的数据；
- 【Import Data】：从工作空间或者一个文件中导入数据；
- 【Export Data】：将训练数据导出到工作空间或者一个文件中；
- 【Train Network】：开始网络模型的训练，训练前必须已经产生或者导入了数据；
- 【OK】、【Apply】：在网络模型经过训练后，单击这两个按钮中的任一个都可以将网络导入 Simulink 模型；
- 【Cancel】：取消刚才的设置。

如上所述，【Generate Training Data】按钮用于产生网络的训练样本，单击该按钮，MATLAB 通过 Simulink 网络模型产生一系列的随机阶跃信号，作为网络的训练样本，如图 7-23 所示。

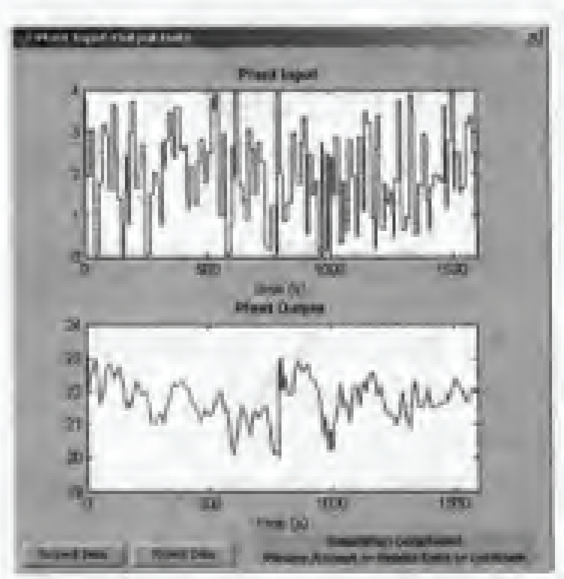


图 7-23　网络训练样本

图 7-23 中有两个按钮：【Accept Data】和【Refuse Data】，前者用于接收数据，如果认为这些数据合理，就单击这个按钮；否则，就单击后者，重新生成训练样本。这里认为这些数据是合理的，所以单击【Accept Data】按钮，然后在系统辨识参数设置窗口中单击【Train Network】按钮，网络开始训练，训练的时间和过程与所选择的训练函数有关，默认的训练函数为 trainlm，训练误差曲线如图 7-24 所示，模型训练数据如图 7-25 所示，模型检验数据如图 7-26 所示。

此时，可以单击【Train Network】按钮，利用同样的数据对网络进行继续训练，也可以在系统辨识参数设置窗口中单击【Erase Generated Data】按钮，擦除现有的训练数据，并且按照以上步骤重新生成训练数据对网络进行训练。当然，也可以认为该模型已经非常精确了，从而接受这个模型。

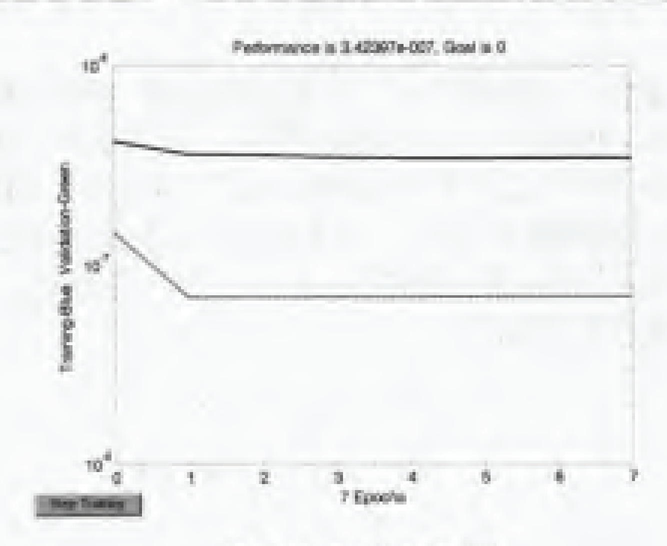


图 7-24　训练误差曲线

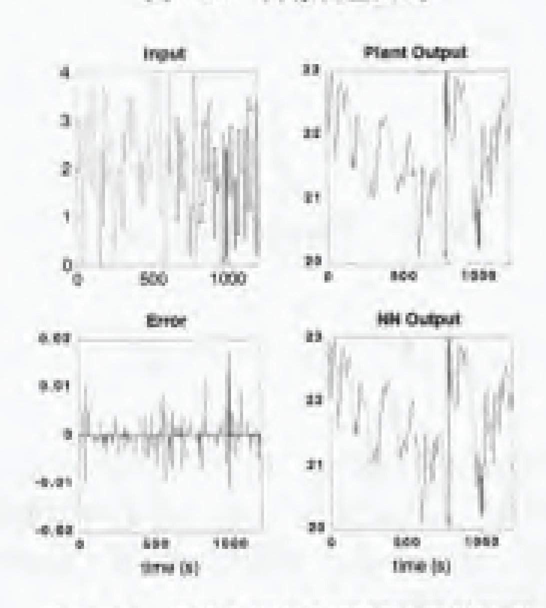


图 7-25　网络预测控制所需的训练数据

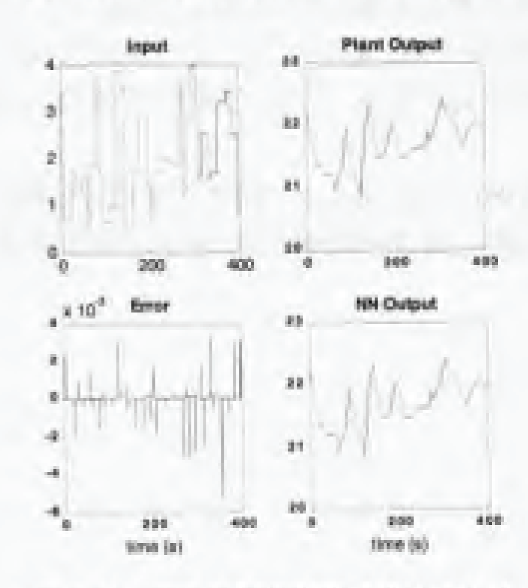


图 7-26　网络预测控制所需的检验数据



4. 仿真

在接受训练好的模型后，回到如图 7-22 所示的系统辨识参数设置窗口，单击【OK】按钮，将该模型导入到 NN Predictive Controller 模块中。在图 7-21 的控制器参数设置窗口中单击【OK】按钮，将控制器参数导入到 NN Predictive Controller 模块中。

返回到如图 7-19 所示的控制模型窗口，在【Simulation】菜单中输入 Start 命令开始仿真，仿真结果如图 7-27 所示，图中的阶梯信号表示参考信号，不规则的曲线表示系统输出，由此可见，控制器的性能还是不错的。

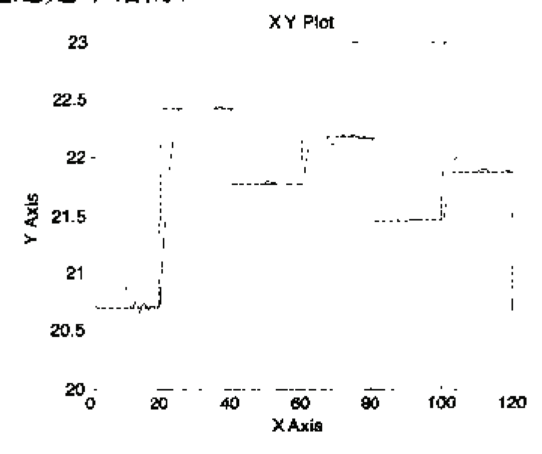


图 7-27　参数信号及系统输出

7.4 小　　结

本章主要介绍了两种神经网络控制系统：模型预测控制系统和反馈线性化控制系统。首先利用 MATLAB 的神经网络工具箱函数和 MATLAB 的脚本编程功能，针对一个简单的非线性系统，实现了反馈线性化控制；然后利用神经网络工具箱自带的演示系统，详细介绍并分析了基于神经网络模型预测控制。

将基于神经网络的模型预测控制策略应用到水反应器中，实现了对产品浓度的有效控制。通过比较，基于神经网络控制系统的控制品质和鲁棒性都优于传统的浓度控制系统。基于神经网络的模型预测控制策略为具有延迟和惯性环节的工业控制对象的控制系统设计提供了一种新的思路。

对于反馈线性化控制来说，首先建立一个神经网络系统模型。接着，使用这个系统模型来训练一个神经网络控制器，迫使系统输出跟踪参考模型的输出。这种控制器的在线计算时间最少。

第8章 基于神经网络的故障诊断

随着现代科学技术水平的日益提高，尤其是计算机科学和控制科学的飞速发展，使得系统的规模和复杂程度迅速增加，设备的安全性和可靠性问题越来越突出。安全保障已经逐渐成为系统运行的一个重要组成部分，系统中出现的某些微小故障若不能及时检测并排除，就有可能造成整个系统的失效、瘫痪，甚至导致巨大的灾难性后果。因此，人们总是期望建立一套监测、预警、容错和维修机制，伴随系统运行的全寿命周期，防止和杜绝影响系统正常运行的故障的发生和发展。

国内外大量的文献资料表明，在实际需求的牵引下，故障诊断技术的应用领域越来越广泛，已经从传统的机械系统和电子系统，渗透到机电一体化系统、工业自动化系统、计算机系统，以及各种广泛意义上的动态系统，包括目标识别系统、组合导航系统等。随着物理学、数学等基础科学的不断进步，以及控制理论、信息科学等应用科学的不断发展，为故障诊断提供了多种技术手段，成为故障诊断技术发展的推动力量。因此，系统故障诊断技术越来越呈现出更宽泛、更深入和更有效的发展态势。

神经网络技术的出现，为故障诊断问题提供了一种新的解决途径，特别是对于在实际中难以建立数学模型的复杂系统，神经网络更显示出其独特的作用。总的来说，神经网络之所以可以成功地应用于故障诊断领域，主要基于以下3个方面的原因：

（1）训练过的神经网络能存储有关过程的知识，能直接从历史故障信息中学习。可以根据对象的日常历史数据训练网络，然后将此信息与当前测量数据进行比较，以确定故障的类型。

（2）神经网络具有滤除噪声及在有噪声情况下得出正确结论的能力，可以训练人工神经网络来识别故障信息，使其能在噪声环境中有效地工作，这种滤除噪声的能力使得人工神经网络适合在线故障检测和诊断。

（3）神经网络具有分辨故障原因及故障类型的能力。

本章主要内容有：

- 神经网络与故障模式识别
- 基于BP网络和Elman网络的齿轮箱故障诊断
- 基于SOM网络的回热系统故障诊断
- 基于概率神经网络的故障诊断
- 基于BP网络的设备状态分类器设计
- 基于RBF网络的船用柴油机故障诊断

8.1 神经网络与故障模式识别

广义地讲，故障可以理解为系统的任何异常现象，使系统表现出所不期望的特性，通常表现为系统的某些（个）重要变量或特性偏离了正常范围。人们对故障的认识起初

是通过选择敏感特性和进行简单比较实现的，这对于简单系统容易做到，而对于复杂系统和复杂现象，就涉及到故障模式和正常模式的识别问题，模式建立及其识别的复杂性主要取决于系统的复杂性和人们的认识水平。人们会通过获取各种先验信息，建立设备正常/故障，以及各种不同故障的样板模式。故障诊断时，根据不同的故障征兆完成模式映射过程。

8.1.1 常用的模式识别方法

自然界的事物和现象一般可分为多个相似，但又不完全相同的群体或个体组成的类别，人们把这样的类别称为模式类或模式，而把其中每个事物或现象称为该模式的一个样本。同类的样本彼此相似，具有某些共同的特征，不同类的样本彼此互不相似。所谓模式识别就是从模式空间到类别隶属空间的正确映射。

故障诊断中经常用到以下模式识别方法：

（1）统计分类方法。该方法是利用了各模式类的分布特征，即直接利用各类的概率密度函数、后验概率等，或隐含地利用上述概念进行分类识别。按照判别准则来划分统计分类方法，包括最小误判概率准则和最小损失判决规则等。

（2）聚类分类方法。为了避免估计概率密度的困难，可以采用该方法。在一定条件下，根据样本空间的相似性把样本集分为若干子集，结果应是某种表示聚类质量的准则函数为最大。常用样本的相似性测度包括距离指标和角度指标。聚类分类方法是一种无监督的学习方法，就是不利用样本的类别属性知识，只根据样本的相似性进行分类的方法。这种方法的前提是，同类样本的特征向量相互靠近而不同类样本的特征向量距离要大得多。常用的方法包括 C-均值法和 ISODATA 算法。

（3）模糊模式识别。该方法利用模糊数学的理论和方法来解决模式识别问题，因此适用于分类识别对象或要求的识别结果具有模糊性的场合。目前，模糊模式识别的方法很多，最简单、最常用的就是最大隶属度原则。

在传统的模式识别技术中，模式分类的基本方法是利用判别函数来划分每个类别。在很多情况下，特别是对于线性不可分的复杂决策区域，判别函数的形式也就格外复杂。而且由于全面的典型参考模式样本是不容易得到的，但如果采用概率模型，会损失模式识别的精度。

8.1.2 神经网络在故障模式识别中的应用

神经网络作为一种自适应的模式识别技术，并不需要预先给出有关模式的经验知识和判别函数，它通过自身的学习机制自动形成所要求的决策区域。网络的特性由其拓扑结构、神经元特性、学习和训练规则所决定。它可以充分利用状态信息，对来自于不同状态的信息逐一进行训练而获得某种映射关系。而且网络可以连续学习，如果环境发生改变，这种映射关系还可以自适应地进行调整。

因此，神经网络由于自身的特性，在故障模式识别领域中有着越来越广泛的应用。下面以单隐层 BP 网络为例，介绍基于神经网络的故障诊断的方法和特点。其中，网络的输



入结点对应着故障征兆，输出结点对应着故障原因。首先利用一组故障样本对网络进行训练，以确定网络的结构（中间层的传递函数和神经元数目）和参数（神经元之间的连接权值和阈值）。网络训练完毕后，故障的模式分类就是根据给定的一组征兆，实现征兆集到故障集之间的非线性映射的过程。

利用神经网络进行故障模式识别具有以下特点：

（1）可用于系统模型未知或系统模型较为复杂，以及非线性系统的故障模式识别。

（2）兼有故障信号的模式变换和特征提取功能。

（3）对系统含有不确定因素、噪声及输入模式不完备的情况下不太敏感。

（4）可用于复杂多模式的故障诊断。

（5）可用于离线诊断，也能适应实时监测的要求。

典型的基于神经网络模式识别功能的诊断系统结构如图 8-1 所示。

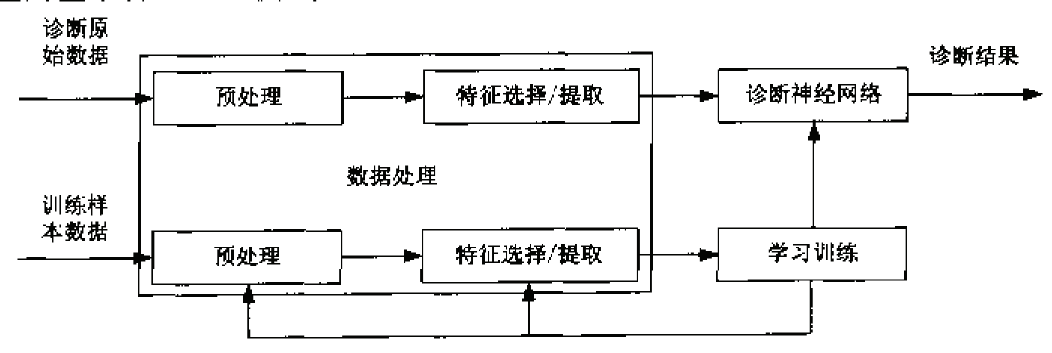


图 8-1　基于神经网络模式识别功能的诊断系统结构

图 8-1 中，基于神经网络的诊断过程分为两步。首先，基于一定数量的训练样本集（通常称为“征兆—故障”数据集）对神经网络进行训练，得到期望的诊断网络；其次，根据当前诊断输入对系统进行诊断，诊断的过程即为利用神经网络进行前向计算的过程。在学习和诊断之前，通常需要对诊断原始数据和训练样本数据进行适当的处理，包括预处理和特征选取/提取等，目的是为诊断网络提供合适的诊断输入和训练样本。此外，尽管神经网络和传统的故障诊断是两种不同的诊断方法，但两者是紧密联系在一起的。如采用小波分析等数据处理方法，可以为神经网络诊断提供可以利用的特征向量。

前向 BP 网络和 RBF 网络的学习算法属于有教师型的。这种算法模型具有很好的推广能力，用于故障模式识别的效果比较好。训练好的 BP 网络和 RBF 网络计算速度快、内存消耗低、可用于实时监测和诊断。但是这两种模型要求学习样本具有一定的致密性、遍历性和相容性，在实际工程中，有时候获得这样的样本比较困难。利用 BP 网络进行故障诊断的一般步骤和注意事项如下：

（1）确定合理的网络结构和规模，尤其是网络中间层神经元个数的选择是网络结构确定和网络性能的关键。

（2）确定训练样本集和测试集。训练样本集用于对网络进行训练，而测试集用于监测网络训练的效果和推广能力。一般来说，训练样本集不仅应全面涵盖所有故障模式类的数据，还应具有一定的代表性，同时还必须保证学习的有效性。测试样本集的选择应该满足“交叉检验（cross validation）”的原则。

（3）根据训练样本集对网络进行训练，经过测试的训练结果即为神经网络故障诊断知

识库。

（4）根据诊断输入，利用 BP 网络进行诊断。

ART 网络和 SOM 网络均属于无教师的竞争学习的自组织网络。ART 网络可以在线学习，边学习边记忆。给网络提供一组样本，它自动形成一组分类模式。如果一个新的输入不能归入任何一个已经形成的模式类中，网络又自动生成一个新的模式类，同时如果新的输入在已经形成的模式类中可以找到一个相似的类，那么这个输入归入该模式类，且网络向更接近这个输入的方向进行调整。SOM 网络采用离线的方式进行学习，它能够很好地进行特征提取，适用于用做最邻近分类器。目前，这两类网络在故障诊断中应用还比较少。虽然这类模型的推广性能没有 BP 网络和 RBF 网络的好，但它们具有的自组织和自适应特性为在复杂系统故障诊断中的应用奠定了坚实的基础。

8.2 基于 BP 网络和 Elman 网络的齿轮箱故障诊断

本节以某型号拖拉机的齿轮箱为工程背景，利用 MATLAB 的神经网络工具箱，介绍基于 BP 网络和 Elman 网络进行齿轮箱故障诊断的过程。

8.2.1 工程描述

拖拉机变速箱是整机进行减速增扭的部件，它受扭转和拉压两种载荷的综合作用，受力过程非常复杂。因此，拖拉机的很多故障出现于变速箱中齿轮及传动轴等机械系统中。据统计，以齿轮为代表的变速箱故障发生率占据除发动机故障以外的其他所有故障的 59%～70%。在非拆卸状态下，传统的齿轮箱故障诊断手段往往依赖于专家的经验判断。但是，由于齿轮箱是一种非常复杂的传动机构，它的故障模式和特征量之间是一种非常复杂的非线性关系，再加上齿轮箱在不同工况下的随机因素，所以专家的经验并不能解决所有的诊断问题。而应用神经网络可以有效地避免这个问题。神经网络的自适应、自学习和对非线性系统超强的分析能力注定它可以在齿轮箱的故障诊断中大显身手。

8.2.2 输入和目标向量设计

神经网络输入的确定实际上就是特征量的提取，对于特征量的选取，主要考虑它是否与故障有比较确定的因果关系，如果输入/输出征兆参数和故障没有任何关系，就不能建立它们之间的联系。

统计表明，齿轮箱故障中有 60%左右都是由齿轮故障导致的，所以这里只研究齿轮故障的诊断。对于齿轮的故障，这里选取了频域中的几个特征量。频域中齿轮故障比较明显的是在啮合频率处的边缘带上。所以，在频域特征信号的提取中选取了在 2、4、6 档时，在 1、2、3 轴的边频带族 $f_s \pm nf_z$ 处的幅值 $A_{i,j1}$、$A_{i,j2}$ 和 $A_{i,j3}$，其中 f_s 表示齿轮的啮合频

率，f_z 是轴的转频，n=1,2,3，i=2,4,6 表示档位，j=1,2,3 表示轴的序号，由于在 2 轴和 3 轴上有两对齿轮啮合，所以用 1、2 分别表示两个啮合频率。这样一来，网络的输入就是一个 15 维的向量，这些数据具有不同的单位和量级，所以在输入神经网络之前应该首先进行归一化处理。表 8-1 给出了输入向量的 9 组数据，它们都是已经归一化后的样本数据。

表 8-1　齿轮箱状态样本数据

数据序号	特征样本	齿轮状态
1	0.2286 0.1292 0.0720 0.1592 0.1335 0.0733 0.1159 0.0940 0.0522 0.1345 0.0090 0.1260 0.3619 0.0690 0.1828	无故障
2	0.2090 0.0947 0.1393 0.1387 0.2558 0.0900 0.0771 0.0882 0.0393 0.1430 0.0126 0.1670 0.2450 0.0508 0.1328	无故障
3	0.0442 0.0880 0.1147 0.0563 0.3347 0.1150 0.1453 0.0429 0.1818 0.0378 0.0092 0.2251 0.1516 0.0858 0.0670	无故障
4	0.2603 0.1715 0.0702 0.2711 0.1491 0.1330 0.0968 0.1911 0.2545 0.0871 0.0060 0.1793 0.1002 0.0789 0.0909	齿根裂纹
5	0.3690 0.2222 0.0562 0.5157 0.1872 0.1614 0.1425 0.1506 0.1310 0.0500 0.0078 0.0348 0.0451 0.0707 0.0880	齿根裂纹
6	0.0359 0.1149 0.1230 0.5460 0.1977 0.1248 0.0624 0.0832 0.1640 0.1002 0.0059 0.1503 0.1837 0.1295 0.0700	齿根裂纹
7	0.1759 0.2347 0.1829 0.1811 0.2922 0.0655 0.0774 0.2273 0.2056 0.0925 0.0078 0.1852 0.3501 0.1680 0.2668	断齿
8	0.0724 0.1909 0.1340 0.2409 0.2842 0.0450 0.0824 0.1064 0.1909 0.1586 0.0116 0.1698 0.3644 0.2718 0.2494	断齿
9	0.2634 0.2258 0.1165 0.1154 0.1074 0.0657 0.0610 0.2623 0.2588 0.1155 0.0050 0.0978 0.1511 0.2273 0.3220	断齿

接下来确定网络的输出模式，由于齿轮包括 3 种故障模式，因此可以采用如下的形式来表示输出：

无故障：（1，0，0）；

齿根裂纹：（0，1，0）；

断齿：（0，0，1）。

8.2.3　BP 网络设计

这里采用 BP 网络进行故障诊断。

1．网络创建

BP 网络模型结构的确定有两条比较重要的指导原则：

（1）对于一般的模式识别问题，三层网络可以很好地被解决。

（2）三层网络中，隐含层神经元个数 n_2 和输入层神经元个数 n_1 之间有以下近似关系：

$$n_2 = 2n_1 + 1$$

由此，可按照如下的方式设计网络，网络的输入层神经元个数为15个，输出层神经元个数为3个，隐含层的神经元个数近似为31个。隐含层的神经元个数并不是固定的，需要经过实际训练的检验来不断调整。

可利用以下代码来创建刚刚设计的网络。网络的输入向量范围为[0,1]，隐含层神经元的传递函数采用S型正切函数tansig，输出层神经元传递函数采用S型对数函数logsig，这是由于输出模式为0－1，正好满足网络的输出要求。

```
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[31,3],{'tansig','logsig'},'trainlm');
```

其中，变量threshold定义了输入向量的最大值和最小值。网络参数如表8-2所示。

表8-2　网络参数

训练函数	学习函数	性能函数
trainlm	learngdm	mse

2．网络训练与测试

网络训练过程是一个不断修正权值和阈值的过程，通过调整，使网络的输出误差达到最小，满足实际应用的要求。

训练函数trainlm是利用Levenberg-Marquardt算法对网络进行训练的，通过以下代码调用trainlm。网络的训练参数设置如表8-3所示。

```
net.trainParam.epochs=1000;
net.trainParam.goal=0.01;
LP.lr=0.1;
net=train(net,P,T);
```

其中，P和T分别为网络的输入向量和目标向量，P是从表8-1中得出的。

表8-3　训练参数

训练次数	训练目标	学习速率
1000	0.01	0.1

网络训练结果为：

```
TRAINLM, Epoch 0/1000, MSE 0.405717/0.01, Gradient 0.597169/1e-010
TRAINLM, Epoch 25/1000, MSE 0.11111/0.01, Gradient 6.96188e-005/1e-010
TRAINLM, Epoch 29/1000, MSE 0.00622753/0.01, Gradient 0.778903/1e-010
TRAINLM, Performance goal met
```

可见，经过29次训练后，网络的性能就达到了要求，如图8-2所示。收敛速度快的一个重要原因在于学习速率的设定值比较大。

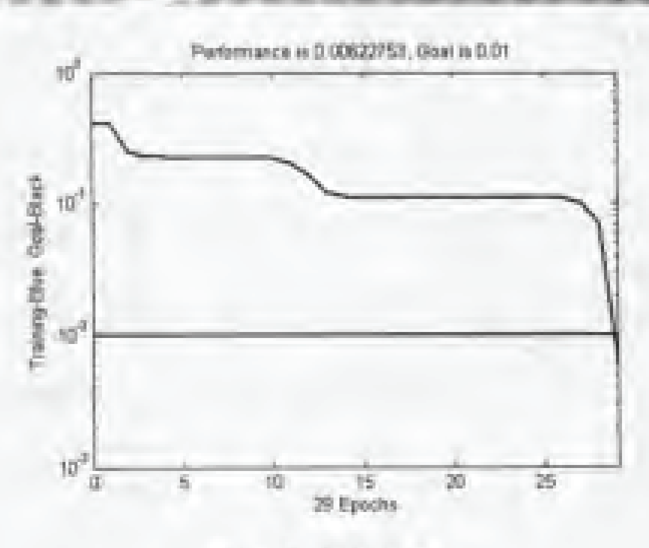


图 8-2　训练结果

接下来需要对训练好的网络进行测试。抽取 3 组新的数据作为网络的测试输入数据，如表 8-4 所示。

表 8-4　测试数据

数据序号	特征样本	齿轮状态
10	0.2101 0.0950 0.1298 0.1359 0.2601 0.1001 0.0753 0.0890 0.0389 0.1451 0.0128 0.1590 0.2452 0.0512 0.1319	无故障
11	0.2593 0.1800 0.0711 0.2801 0.1501 0.1298 0.1001 0.1891 0.2531 0.0875 0.0058 0.1803 0.0992 0.0802 0.1002	齿根裂纹
12	0.2599 0.2235 0.1201 0.1171 0.1102 0.0683 0.0621 0.2597 0.2602 0.1167 0.0048 0.1002 0.1521 0.2281 0.3205	断齿

测试代码为：

```
Y=sim(net,P_test);
```

测试结果为：

```
Y=
    0.9618    0.0075    0.0103
    0.0000    0.7658    0.0000
    0.0002    0.0000    0.9483
```

按照欧式范数理论，这 3 次测试的误差分别为 0.0382、0.2343 和 0.0527，可以看出，这些误差是非常小的。因此，可以判定，经过训练后，网络是完全可以满足齿轮箱故障诊断的要求的。

本实例的完整 MATLAB 代码如下：

```
P=[0.2286 0.1292 0.0720 0.1592 0.1335 0.0733 0.1159 0.0940 0.0522 0.1345 0.0090 0.1260
  0.3619 0.0690 0.1828;
  0.2090 0.0947 0.1393 0.1387 0.2558 0.0900 0.0771 0.0882 0.0393 0.1430 0.0126 0.1670 0.2450
  0.0508 0.1328;
  0.0442 0.0880 0.1147 0.0563 0.3347 0.1150 0.1453 0.0429 0.1818 0.0378 0.0092 0.2251 0.1516
  0.0858 0.0670;
  0.2603 0.1715 0.0702 0.2711 0.1491 0.1330 0.0968 0.1911 0.2545 0.0871 0.0060 0.1793 0.1002
```

```
        0.0789 0.0909;
        0.3690 0.2222 0.0562 0.5157 0.1872 0.1614 0.1425 0.1506 0.1310 0.0500 0.0078 0.0348 0.0451
        0.0707 0.0880;
        0.0359 0.1149 0.1230 0.5460 0.1977 0.1248 0.0624 0.0832 0.1640 0.1002 0.0059 0.1503 0.1837
        0.1295 0.0700;
        0.1759 0.2347 0.1829 0.1811 0.2922 0.0655 0.0774 0.2273 0.2056 0.0925 0.0078 0.1852 0.3501
        0.1680 0.2668;
        0.0724 0.1909 0.1340 0.2409 0.2842 0.0450 0.0824 0.1064 0.1909 0.1586 0.0116 0.1698 0.3644
        0.2718 0.2494;
        0.2634 0.2258 0.1165 0.1154 0.1074 0.0657 0.0610 0.2623 0.2588 0.1155 0.0050 0.0978 0.1511
        0.2273 0.3220]';
T=[1 0 0;1 0 0;1 0 0;
   0 1 0;0 1 0;0 1 0;
   0 0 1;0 0 1;0 0 1]';
%输入向量的最大值和最小值
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[31,3],{'tansig','logsig'},'trainlm');
%训练次数为 1000，训练目标为 0.01，学习速率为 0.1
net.trainParam.epochs=1000;
net.trainParam.goal=0.01;
LP.lr=0.1;
net=train(net,P,T);
%测试数据，和训练数据不一致
P_test=[0.2101 0.0950 0.1298 0.1359 0.2601 0.1001 0.0753 0.0890 0.0389 0.1451 0.0128 0.1590
        0.2452 0.0512 0.1319;
        0.2593 0.1800 0.0711 0.2801 0.1501 0.1298 0.1001 0.1891 0.2531 0.0875 0.0058 0.1803
        0.0992 0.0802 0.1002;
        0.2599 0.2235 0.1201 0.1171 0.1102 0.0683 0.0621 0.2597 0.2602 0.1167 0.0048 0.1002
        0.1521 0.2281 0.3205]';
Y=sim(net,P_test)
```

8.2.4 Elman 网络设计

BP 网络有着很强的非线性映射能力，在故障诊断中的应用也非常成功。但是，BP 网络是一种前向的神经网络，相对于反馈型的网络来说，收敛速度相对较慢，而且有可能收敛到局部极小点。因此，这里尝试利用 Elman 网络再次对上述实例进行故障诊断，并比较两者的结果。

1．网络创建

由于单隐层的 Elman 网络的功能已经非常强大，因此，这里采用单隐层的网络就足够了。最影响网络性能的是隐含层神经元的个数，而这又是比较难以确定的。

由于输入向量的维数为 15，因此输入层神经元的个数应该为 15；而输出向量的维数为 3，则输出层神经元的个数就为 3。综合考虑网络的性能和速度，将隐含层神经元的个数设定为 25。利用以下代码创建一个 Elman 网络：

```
net=newelm(minmax(P),[25,3],{'tansig','logsig'})
```

其中，P 和 T 分别为输入向量和目标向量。

2. 网络训练及应用

利用以下代码对网络进行训练：

```
net.trainParam.epochs=500;
net.trainParam.goal=0.01;
net=train(net,P,T);
```

训练结果为：

```
TRAINGDX, Epoch 0/500, MSE 0.341503/0.01, Gradient 0.155039/1e-006
TRAINGDX, Epoch 25/500, MSE 0.33327/0.01, Gradient 0.14442/1e-006
TRAINGDX, Epoch 50/500, MSE 0.313313/0.01, Gradient 0.115666/1e-006
TRAINGDX, Epoch 75/500, MSE 0.278458/0.01, Gradient 0.0760164/1e-006
TRAINGDX, Epoch 100/500, MSE 0.225498/0.01, Gradient 0.0546082/1e-006
TRAINGDX, Epoch 125/500, MSE 0.0947014/0.01, Gradient 0.0380976/1e-006
TRAINGDX, Epoch 142/500, MSE 0.00943479/0.01, Gradient 0.0146832/1e-006
TRAINGDX, Performance goal met.
```

可见，经过 142 次训练后，网络误差达到要求。如图 8-3 所示，网络的训练误差曲线比较平滑。

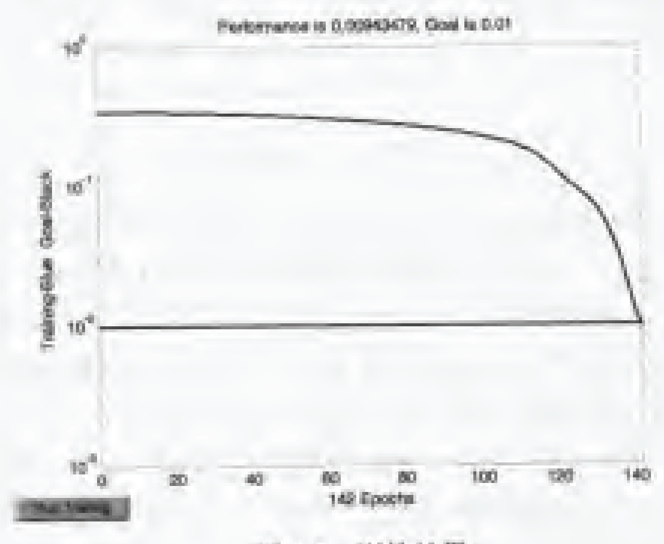


图 8-3 训练结果

利用测试数据 P_test 对网络进行仿真：

```
Y=sim(net,P_test);
```

结果为：

```
Y =
    0.9129    0.0358    0.0046
    0.0941    0.8650    0.1323
    0.0708    0.0784    0.9513
```

可以看出，Elman 网络也准确地识别出了所有的故障类型。相对于 BP 网络来说，Elman 网络的识别误差要大一些，但这并不影响实际应用。另外，由于在 Elman 网络中引入了反馈，所以网络的训练误差曲线要比 BP 网络平滑。

完整的 MATALB 代码为：

```
net=newelm(minmax(P),[25,3],{'tansig','logsig'})
```

```
net.trainParam.epochs=500;
net.trainParam.goal=0.01;
net=train(net,P,T);
Y=sim(net,P_test);
```

8.3 基于 SOM 网络的回热系统故障诊断

回热系统是火力发电机组热力系统中最主要、最复杂的系统，随着机组容量和参数的提高，热力系统特别是回热系统对整个机组的安全性和经济性的影响越来越大。因此，应根据传感器测得的各种状态参数来判断回热系统的运行状态，一旦发生异常就立即进行诊断，及时找出故障原因。

采用自组织特征映射人工神经网络（SOM）能很好地完成这种识别任务。自组织特征映射人工神经网络采用的是无教师学习的自学习方式，无需在训练或学习过程中预先指明这个训练输入矢量的所属类别。当输入某一类别的矢量时，神经网络中的一个神经元将会在其输出端产生最大值，而其他的神经元具有最小输出值。所以，该网络能够根据最大值的神经元的位置来判断输入矢量所代表的故障。本节尝试将自组织特征映射神经网络应用在回热系统的故障诊断中。

8.3.1 背景

根据回热系统的运行经验及现场条件，利用 9 个运行参数提取故障征兆，归纳得到 13 种（正常状态亦作为一种模式）主要故障的样本特征模式，如表 8-5 所示。这里需要特别指出的是，在故障征兆参数的处理上，由于各个参数在系统中具有不同的特点，在发生可能的故障时，它们的变化范围和方向都各不相同。又因为神经网络对输入样本有归一化的要求，因此对参数的变化方向和程度的表示需要进行慎重的处理。

这里根据各个参数的不同特点来表示它们的变化范围和程度。考虑到参数测点的波动，将只能变大或变小的参数的正常值设定为 0.25 和 0.75，而将可能双向变化的参数的正常值设定为 0.5。这样，作为神经网络的训练样本，大多数的参数在归一化后其变化范围加大了，极大地改善了训练的收敛性。

表 8-5　回热系统故障样本特征模式

对应故障	抽气流量	抽气压力	进口压力	进口水温	出口水温	混合水温	出口端差	水　位	疏水温度
A.排气管卡涩	0.25	0.60	0.25	0.50	0.50	0.50	0.25	0.50	0.25
B.排气管不畅	0.25	0.60	0.50	0.50	0.50	0.50	0.75	0.50	0.50
C.排气量过大	0.75	0.40	0.75	0.50	0.50	0.50	0.50	0.50	0.50
D.管束污染	0.25	0.60	0.50	0.50	0.25	0.25	0.75	0.50	0.50
E.水侧短路	0.25	0.60	0.50	0.50	0.00	0.00	1.00	0.50	0.50
F.管束泄漏	0.75	0.40	0.75	0.50	0.25	0.25	0.75	1.00	0.25

（续表）

对应故障	抽气流量	抽气压力	进口压力	进口水温	出口水温	混合水温	出口端差	水　位	疏水温度
G.疏水不畅	0.25	0.60	0.50	0.50	0.50	0.50	0.75	0.75	0.50
H.疏水阀故障	0.75	0.40	0.50	0.50	0.75	0.75	0.25	0.00	1.00
I.旁路故障	0.25	0.60	0.50	0.50	0.75	0.25	0.25	0.50	0.50
J.加热器满水	0.25	0.60	0.50	0.50	0.25	0.25	0.75	1.00	0.50
K.排气带水	0.50	0.50	0.50	0.50	0.75	0.75	0.25	1.00	0.50
L.自身沸腾	0.00	0.75	0.75	0.50	1.00	0.75	0.25	1.00	0.50
M.运行正常	0.50	0.50	0.50	0.50	0.75	0.75	0.25	0.50	0.50

8.3.2　SOM 网络设计

网络输入向量元素的个数为 9 个，范围都在[0,1]之间。为了提高网络映射精度，将网络的竞争层设计为一个 8×8 的二维平面。

```
net=newsom(minmax(P),[8 8]);
```

其中，P 为网络的输入向量。

接下来对网络进行训练：

```
net=train(net,P);
y=sim(net,P);
yc=vec2ind(y)
```

训练结果为：

```
yc=
    29    19    61     3     1    57    35    64    13    41    24     8    40
```

上述结果可以在图 8-4 中表示出来。由图可见，经过 100 次训练后，训练好的网络将所有的故障模式进行了成功的分类。

E		D					L
				I			
		B					K
				A			
		G					M
J							
F				C			H

图 8-4　训练后的 SOM 网络输出

这时，如果输入一个新的故障模式，竞争层的神经元开始竞争，激活与之最为接近的神经元，从而实现正确的分类。一旦故障的类型确定，网络的故障诊断功能也就实现了。

网络竞争层神经元的个数和排列对于网络性能来说是很重要的，如果神经元个数比较

少，可能就无法对输入模式进行正确的分类。还是上面的例子，将竞争层设置为4×4的二维平面阵列，其他设定参数都不变，然后对网络进行训练并仿真。

输出结果为：

```
yc=
     6     3    14     4     8    16     7    13     6    12     1     1    10
```

可见，网络将故障模式A和I分为一类，将故障模式K和L分为一类，所以说，该网络的分类性能是比较差的，只能做初级的分类。

如果出现这种情况，即使调整训练次数，也无法彻底改变网络的分类性能。比如，将网络训练次数设定为2000，这时的分类结果为：

```
yc=
     6     3    14     4     8    16     7    13     6    12     1     1    10
```

与上面的分类结果是一样的，因此，只有调整网络竞争层的结构才可以提高网络的性能。而竞争层结构的调整需要不断地实验。

本实例的MATALB代码为：

```
%输入向量P
P=[
    0.25  0.60 0.25 0.50 0.50 0.50 0.25 0.50 0.25;
    0.25  0.60 0.50 0.50 0.50 0.50 0.75 0.50 0.50;
    0.75  0.40 0.75 0.50 0.50 0.50 0.50 0.50 0.50;
    0.25  0.60 0.50 0.50 0.25 0.25 0.75 0.50 0.50;
    0.25  0.60 0.50 0.50 0.00 0.00 1.00 0.50 0.50;
    0.75  0.40 0.75 0.50 0.25 0.25 0.75 1.00 0.25;
    0.25  0.60 0.50 0.50 0.50 0.50 0.75 0.75 0.50;
    0.75  0.40 0.50 0.50 0.75 0.75 0.25 0.00 1.00;
    0.25  0.60 0.50 0.50 0.75 0.25 0.25 0.50 0.50;
    0.25  0.60 0.50 0.50 0.25 0.25 0.75 1.00 0.50;
    0.50  0.50 0.50 0.50 0.75 0.75 0.25 1.00 0.50;
    0.00  0.75 0.75 0.50 1.00 0.75 0.25 1.00 0.50;
    0.50  0.50 0.50 0.50 0.75 0.75 0.25 0.50 0.50]';
net=newsom(minmax(P),[8 8]);
net=train(net,P);
y=sim(net,P);
yc=vec2ind(y);
%输出聚类结果
yc_88=yc

net=newsom(minmax(P),[8 8]);
net.trainParam.epochs=2000;
net=train(net,P);
y=sim(net,P)
yc=vec2ind(y);
%输出聚类结果
yc_44=yc
```



8.4 基于概率神经网络的故障诊断

概率神经网络 PNN 是一种性能良好的分类神经网络，由于它直接考虑样本空间的概率特性，以样本空间的典型样本作为隐含层的结点，一经确定就不需要进行训练，只要随实际问题进行样本的追加就可以了，而且 PNN 具有全局优化的特点。因此，PNN 已经广泛被应用于故障诊断领域。

8.4.1 概述

概率神经网络 PNN 是一种可用于模式分类的神经网络，其实质是基于贝叶斯最小风险准则发展而来的一种并行算法，目前已经在雷达、心电图仪等电子设备中获得了广泛的应用。PNN 与 BP 网络相比较，其主要优点为：

（1）快速训练，其训练时间仅仅略大于读取数据的时间。

（2）无论分类问题多么复杂，只要有足够多的训练数据，就可以保证获得贝叶斯准则下的最优解。

（3）允许增加或减少训练数据而无需重新进行长时间的训练。

PNN 的结构如图 3-27 所示，这种 PNN 的层次模型是 Specht 根据贝叶斯分类规则与 Parzen 的概率密度函数提出的。在训练网络时，网络直接存储训练样本向量为网络的模式样本向量，而不做任何的修改，只需对高斯函数的平滑因子进行经验式统计的估计，过程极为简单：在网络工作时，待识别样本 X 由输入层直接送到模式层各个类别单元中，在模式单元中进行向量 X 与 W 的点积。完成非线性处理后，再送入求和层中；求和层中各单元只与相应类别的模式单元相连，并且依据 Parzen 方法求和估计各类的概率；在决策层中，根据对输入向量的概率估计，按贝叶斯分类规则将输入向量分到具有最大后验概率值的类别中去。

在进行故障诊断的过程中，求和层将模式层中同一模式的输出求和，并乘以代价因子；决策层则选择求和层中输出最大者对应的故障模式为诊断结果。当故障样本的数量增加时，模式层的神经元将随之增加。而当故障模式多于两种时，则求和层神经元将增加。所以，随着故障经验知识的积累，概率神经网络可以不断横向扩展，故障诊断的能力也将不断提高。

8.4.2 基于 PNN 的故障诊断

1. 问题描述

发动机运行过程中，油路和气路出现故障是最多的。由于发动机结构复杂，很难分清故障产生的原因，所以接下来尝试利用 PNN 来实现发动机的故障诊断。

在发动机运行中常选用的 6 种特征参数为 AI、MA、DI、MD、TR 和 PR。其中，AI 为最大加速度指标，MA 为平均加速度指标，DI 为最大减速度指标，MD 为平均减速度指标，TR 为扭矩谐波分量比，PR 为燃爆时上升速度。

进行诊断时，首先要提取有关的特征参数，然后利用 PNN 进行诊断，诊断模型如图 8-5 所示。

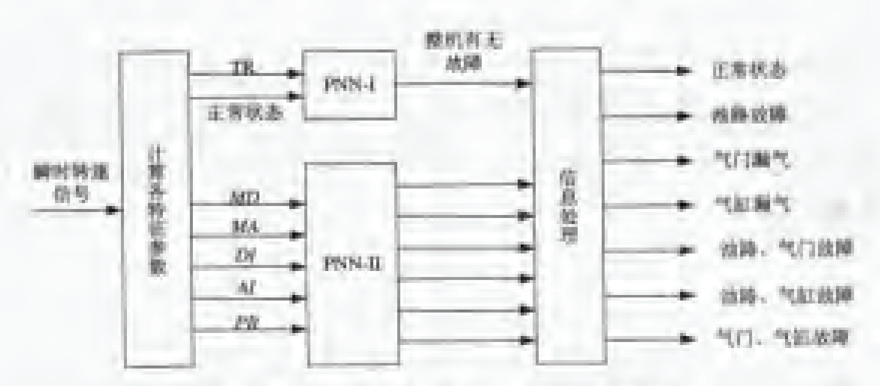


图 8-5 基于 PNN 的发动机诊断模型

2. PNN 的创建和应用

由图 8-5 可见，我们设计了两个 PNN 进行故障诊断。PNN-I 的输入层有两个结点，对应 **TR** 和正常状态；样本模式层有两个结点，分别对应正常和故障两种模式；输出层有两个结点，分别对应正常和故障两种状态。

PNN_II 的输入层有 5 个结点，分别对应 5 个特征参数为 AI、MA、DI、MD、PR；模式层有 10 个结点，对应每个结点的正常和故障中的 10 组模式；输出层有 4 个结点，分别对应油路故障、气门漏气、气缸漏气和正常 4 种状态。所谓信息处理就是通过输出的 4 种状态综合判定实际输出究竟是单故障还是复合故障。

由于信息处理策略不属于本书的讨论范围，因此，这里主要介绍如何利用 PNN 进行故障诊断。限于篇幅的原因，这里选用发动机中的 1 号气缸进行分析。经过分析，该气缸一共出现了 3 种故障，分别为油量少、气门漏气和气缸漏气，再加上正常状态，可以认为一共有 4 种故障模式。利用二进制格式描述这 4 种故障模式，如表 8-6 所示。这 4 种故障模式通过现场试验和对历史资料的收集分析，可以得到 4 组故障样本数据，如表 8-7 所示。

表 8-6 故障模式分类

故障模式	对应描述
正常	1 0 0 0
油量少	0 1 0 0
气门漏气	0 0 1 0
气缸漏气	0 0 0 1

表 8-7 故障样本数据

气缸状态	TR	PR	AI	MA	DI	MD
正常	0.999	0.999	0.999	0.999	0.999	0.999
油量少	0.102	0.091	0.035	0.085	0.078	0.105
气门漏气	0.111	0.125	0.021	0.078	0.121	0.192
气缸漏气	0.215	0.185	0.152	0.098	0.115	0.113

由于这些数据之间相差都不大，因此，不需要进行归一化处理就可以直接应用了。利用这些故障信息作为网络的训练样本，从而创建一个概率神经网络用于故障诊断。PNN 的创建方法和 RBF 网络的创建方法非常相似，代码为：

```
net=newpnn(P,T,SPREAD)
```

其中，P 和 T 分别为输入向量和目标向量，SPREAD 为径向基函数的分布密度，默认为 0.1。为了更好地分析 SPREAD 对网络性能的影响，这里将 SPREAD 设置为 5 个值，分别为 0.1、0.2、0.3、0.4 和 0.5。

函数 newpnn 已经创建了一个准确的概率神经网络，可以利用该网络进行故障诊断和分析了。

首先，检验网络对训练数据的分类：

```
y=sim(net,P);
yc=vec2ind(y);
```

不同的 SPREAD 值对应的概率神经网络的输出结果都是一样的，即：

```
yc =
     1     2     3     4
```

由此可见，网络成功地将故障模式分为了 4 类。为了检验网络的外推性能，接下来给出一组测试样本数据，如表 8-8 所示。这组数据都来源于真实的故障信息，可以有效地检验网络的性能。

表 8-8 测试样本数据

气缸状态	TR	PR	AJ	MA	DI	MD
正常	1.001	1.002	0.999	1.000	1.006	0.998
油量少	0.110	0.102	0.031	0.091	0.082	0.096
气门漏气	0.115	0.131	0.019	0.083	0.123	0.202
气缸漏气	0.221	0.193	0.148	0.105	0.121	0.110

利用上表中的测试样本对网络进行测试，代码为：

```
y_test=sim(net,P_test);
yc_test=vec2ind(y_test);
```

输出结果为：

```
yc_test=
     1     2     3     4
```

由此可见，网络的分类结果是正确的。也就是说，网络成功地诊断出了这 4 种故障，因此，网络用于故障诊断是有效的。

8.4.3 结论

基于概率神经网络的故障诊断方法可以最大程度地利用故障先验知识，在贝叶斯最小风险准则下对发动机的单故障进行定性诊断。概率神经网络训练速度快，在工程上易于实现，而且对样本噪声具有较强的鲁棒性，可以达到较高的诊断准确率。随着故障知识的逐渐积累，网络可以不断扩张从而进一步提高诊断准确率。

本实例的完整 MATLAB 代码为：

```
%输入向量P和目标向量T
P=[0.999   0.999 0.999 0.999 0.999 0.999;
   0.102   0.091 0.035 0.085 0.078 0.105;
   0.111   0.125 0.021 0.078 0.121 0.192;
   0.215   0.185 0.152 0.098 0.115 0.113]';
T=[1 0 0 0;
   0 1 0 0;
   0 0 1 0;
   0 0 0 1]';
%创建5个PNN，SPREAD不同
for i=1:5
      net=newpnn(P,T,i/10);
      temp=sim(net,P)
      yc=vec2ind(temp)
end
%测试样本
P_test=[1.001     1.002 0.999 1.000 1.006 0.998;
        0.110     0.102 0.031 0.091 0.082 0.096;
        0.115     0.131 0.019 0.083 0.123 0.202;
        0.221     0.193 0.148 0.105 0.121 0.110]';
y_test=sim(net,P_test);
yc_test=vec2ind(y_tets);
yc_test
```

8.5 基于 BP 网络的设备状态分类器设计

本实例的工程背景是某设备中的减速箱，其目的是设计一个状态分类器，用于检测减速箱的当前状态。为了简单起见，将减速箱状态分为正常状态、轻微故障状态和严重故障状态等 3 种类别。

8.5.1 BP 网络设计

首先，需要获得网络的输入和目标样本。对减速箱运行状态进行监测，获得了 12 组状态样本数据，有正常状态数据、轻微故障状态数据和严重故障状态数据，分别对应类别 1、2 和 3，如表 8-9 所示。

表 8-9 减速箱状态样本数据

样本序号	样本输入特征数据	类别
1	–1.7817 –0.2786 –0.2954 –0.2394 –0.1842 –0.1572 –0.1584 –0.1998	1
2	–1.8710 –0.2957 –0.3494 –0.2904 –0.1460 –0.1387 –0.1492 –0.2228	1
3	–1.8347 –0.2817 –0.3566 –0.3476 –0.1820 –0.1435 –0.1778 –0.1849	1

（续表）

样本序号	样本输入特征数据	类 别
4	−1.8807 −0.2467 −0.2316 −0.2419 −0.1938 −0.2103 −0.2010 −0.2533	1
5	−1.4151 −0.2282 −0.2124 −0.2147 −0.1271 −0.0680 −0.0872 −0.1684	2
6	−1.2879 −0.2252 −0.2012 −0.1298 −0.0245 −0.0390 −0.0762 −0.1672	2
7	−1.5239 −0.1979 −0.1094 −0.1402 −0.0994 −0.1394 −0.1673 −0.2810	2
8	−1.6781 −0.2047 −0.1180 −0.1532 −0.1732 −0.1716 −0.1851 −0.2006	2
9	0.1605 −0.0920 −0.0160 0.1246 0.1802 0.2087 0.2234 0.1003	3
10	0.2045 0.1078 0.2246 0.2031 0.2428 0.2050 0.0704 0.0403	3
11	−1.0242 −0.1461 −0.1018 −0.0778 −0.0363 −0.0476 −0.0160 −0.0253	3
12	−0.7915 −0.1018 −0.0737 −0.0945 −0.0955 0.0044 0.0467 0.0719	3

接下来需要确定网络结构，根据 Kolmogorov 定理，采用一个 $N\times 2N+1\times M$ 的 3 层 BP 网络作为状态分类器。其中，N 表示输入特征向量的分量数，M 表示输出状态类别总数。对于本实例，N=8。为了简化网络结构，用（0,1）表示正常状态，（1,0）表示轻微故障状态，（1,1）表示严重故障状态。这样一来，就可以在网络中只设计两个输入神经元表示这 3 种故障状态类别。由此可得，该 BP 网络结构为：输入层有 8 个神经元，中间层有 17 个神经元，输出层有 2 个神经元。按照一般的设计方案，中间层神经元的传递函数为 S 型正切函数，输出层神经元的传递函数为 S 型对数函数，之所以选择 S 型对数函数，是因为该函数为 0-1 函数，正好满足设备状态分类器的输出要求。

令 P 表示网络的输入样本向量，T 表示网络的目标向量。由表 8-9 可得：

```
P=[-1.7817 -0.2786 -0.2954 -0.2394 -0.1842 -0.1572 -0.1584 -0.1998;
  -1.8710 -0.2957 -0.3494 -0.2904 -0.1460 -0.1387 -0.1492 -0.2228;
  -1.8347 -0.2817 -0.3566 -0.3476 -0.1820 -0.1435 -0.1778 -0.1849
   - 1.8807 -0.2467 -0.2316 -0.2419 -0.1938 -0.2103 -0.2010 -0.2533;
   - 1.4151 -0.2282 -0.2124 -0.2147 -0.1271 -0.0680 -0.0872 -0.1684;
   - 1.2879 -0.2252 -0.2012 -0.1298 -0.0245 -0.0390 -0.0762 -0.1672;
   - 1.5239 -0.1979 -0.1094 -0.1402 -0.0994 -0.1394 -0.1673 -0.2810;
   - 1.6781 -0.2047 -0.1180 -0.1532 -0.1732 -0.1716 -0.1851 -0.2006;
  0.1605 -0.0920 -0.0160  0.1246  0.1802  0.2087  0.2234  0.1003;
  0.2045  0.1078 0.2246  0.2031  0.2428  0.2050  0.0704  0.0403;
   - 1.0242 -0.1461 -0.1018 -0.0778 -0.0363 -0.0476 -0.0160 -0.0253;
   - 0.7915 -0.1018 -0.0737 -0.0945 -0.0955  0.0044  0.0467  0.0719];
T=[0 1; 0 1; 0 1; 0 1; 0 1; 1 0; 1 0; 1 0; 1 0; 1 1; 1 1; 1 1;1 1]
```



P 和 T 需要保持列数相等，否则在运行过程中会出现错误。

利用如下代码可以创建一个符合以上要求的 BP 网络。网络的训练函数为 trainlm，该函数的运行速度比较快，对于中大型的网络比较合适。学习函数取默认值 learngdm，性能函数取默认值 mse。其中函数 minmax 设定了输入向量元素的阈值范围。

```
%取 P 和 T 转置，保证输入样本向量和目标向量之间列数相等
P=P';
```

```
T=T';
net=newff (minmax(P),[17,2],{'tansig','logsig'},'trainlm')
```

其中，

```
mimax(P)=
            -1.8807    0.2045
            -0.2957    0.1078
            -0.3566    0.2246
            -0.3476    0.2031
            -0.1938    0.2428
            -0.2103    0.2087
            -0.2010    0.2234
            -0.2810    0.1003。
```

8.5.2 网络训练

网络创建后，并不能直接投入使用，必须经过训练并且达到要求后，才可以作为设备状态分类器使用。网络的训练代码如下：

```
%设定循环次数为 50
net.trainParam.epochs=50;
%训练误差为 0.0001，其余训练参数取默认值
net.trainParam.goal=0.001;
net=train(net,P,T);
```

训练结果为：

```
TRAINLM, Epoch 0/50, MSE 0.538103/0.001, Gradient 2.61789/1e-010
TRAINLM, Epoch 17/50, MSE 0.000247899/0.001, Gradient 0.0160769/1e-010
TRAINLM, Performance goal met.
```

可见，经过 17 次训练后，网络误差达到设定的最小值，结果如图 8-6 所示。

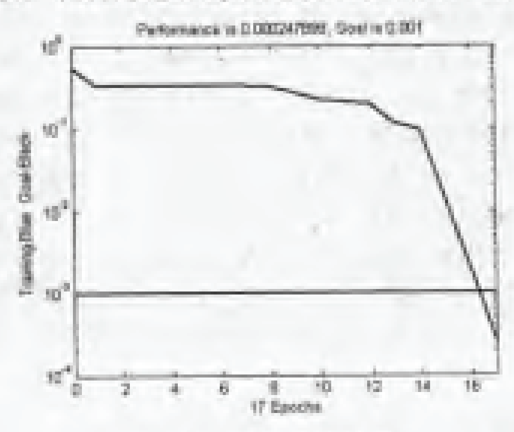


图 8-6　网络训练结果

接下来通过仿真来检验网络的输出是否满足要求，代码如下：

```
sim(net,P)
```

运行结果为：

```
Columns 1 through 6
0.0426    0.0121    0.0220    0.0192    0.9939    0.9976
```

```
0.9898    0.9981    0.9987    0.9910    0.0299    0.0271
Columns 6 through 12
0.9957    0.9881    1.0000    1.0000    0.9869    0.9776
0.0035    0.0193    1.0000    1.0000    0.9926    0.9999
```

对照表 8-9 可知，该网络组成的状态分类器可以有效准确地识别该减速箱已经出现的运行状态，为故障检测提供了有效的工具。

8.5.3 网络测试与应用

网络测试的目的是为了确定网络是否满足实际应用的要求。需要指出的是，测试数据应该和训练用的样本数据不一致，否则，测试得出的结果永远都是满意的。

现在分别在设备正常状态、轻微故障状态和严重故障状态下测得 3 组状态数据，如表 8-10 所示。

表 8-10 测试数据

样本序号	状态数据	类别
13	-1.4736 -0.2845 -3.0724 -0.2108 -0.1904 -0.1467 -0.1696 -0.2001	1
14	-1.6002 -0.2011 -0.1021 -0.1394 -0.1001 -0.1572 -0.1584 -0.2790	2
15	-1.0314 -0.1521 -0.1101 -0.0801 -0.0347 -0.0482 -0.0158 -0.0301	3

试利用上面设计的网络判别它们分别属于哪一种状态。代码如下：

```
P_test=[ - 1.4736  - 0.2845  - 3.0724  - 0.2108  - 0.1904  - 0.1467  - 0.1696  - 0.2001;
         - 1.6002  - 0.2011  - 0.1021  - 0.1394  - 0.1001  - 0.1572  - 0.1584  - 0.2790;
         - 1.0314  - 0.1521  - 0.1101  - 0.0801  - 0.0347  - 0.0482  - 0.0158  - 0.0301]';
Y=sim(net,P_test)
```

运行结果为：

```
Y=
    0.0000    0.9827    0.9873
    1.0000    0.0148    0.9909.
```

由此可见，第 13 组状态数据属于正常状态（0,1），第 14 组状态数据属于轻微故障状态（1,0），第 15 组状态数据属于严重故障状态（1,1）。这与实际情况是相符合的，说明所设计的设备状态分类器是合理的，可以投入实际应用。

本实例的完整 MATLAB 代码如下：

```
P=[-1.7817 -0.2786 -0.2954 -0.2394 -0.1842 -0.1572 -0.1584 -0.1998;
   -1.8710 -0.2957 -0.3494 -0.2904 -0.1460 -0.1387 -0.1492 -0.2228;
-1.8347 -0.2817 -0.3566 -0.3476 -0.1820 -0.1435 -0.1778 -0.1849;
-1.8807 -0.2467 -0.2316 -0.2419 -0.1938 -0.2103 -0.2010 -0.2533;
-1.4151 -0.2282 -0.2124 -0.2147 -0.1271 -0.0680 -0.0872 -0.1684;
-1.2879 -0.2252 -0.2012 -0.1298 -0.0245 -0.0390 -0.0762 -0.1672;
-1.5239 -0.1979 -0.1094 -0.1402 -0.0994 -0.1394 -0.1673 -0.2810;
-1.6781 -0.2047 -0.1180 -0.1532 -0.1732 -0.1716 -0.1851 -0.2006;
0.1605 -0.0920 -0.0160   0.1246   0.1802   0.2087   0.2234   0.1003;
0.2045   0.1078 0.2246   0.2031   0.2428   0.2050   0.0704   0.0403;
-1.0242 -0.1461 -0.1018 -0.0778 -0.0363 -0.0476 -0.0160 -0.0253;
```

```
-0.7915 -0.1018 -0.0737 -0.0945 -0.0955   0.0044   0.0467   0.0719];
T=[0 1;0 1;0 1;0 1;1 0;1 0;1 0;1 0;1 1;1 1;1 1;1 1];
P=P';
T=T';
net=newff(minmax(P),[17,2],{'tansig','logsig'},'trainlm');
net.trainParam.epochs=50;
net.trainParam.goal=0.001;
net=train(net,P,T);

P_test=[-1.4736 -0.2845 -3.0724 -0.2108 -0.1904 -0.1467 -0.1696 -0.2001;
 - 1.6002 –0.2011 –0.1021 –0.1394 –0.1001 –0.1572 –0.1584 –0.2790;
-1.0314 -0.1521 -0.1101 -0.0801 -0.0347 -0.0482 -0.0158 -0.0301]';
Y=sim(net,P_test)
```

8.6 基于 RBF 网络的船用柴油机故障诊断

船用柴油机是整个船舶的动力装置，柴油机一旦发生故障就会对船舶的安全性造成很大的威胁。因此，如何迅速判断故障发生的原因，进而有效地排除故障，保证船舶的继续正常航行具有特别重要的意义。传统的柴油机故障诊断方法包括很多种，如润滑油法、性能参数法和振动噪声法等。

近年来，随着计算机技术、信号分析技术、模糊和人工智能技术的发展，柴油机故障诊断技术得到了很大程度上的发展。尤其是以非线性并行分布处理为主流的神经网络理论的发展，为柴油机故障诊断技术的研究开辟了新的途径。通过专家经验和对柴油机过程的模拟运算，归结出具有典型特征的样本，组织构造相应的神经网络，采用样本集对网络进行训练，使得网络通过学习，把样本中的输入和目标向量之间的对应关系记忆在网络结构上。通过神经网络推理，可以对柴油机进行故障诊断。

本节主要研究如何利用 MATLAB 神经网络工具箱，基于 RBF 网络进行柴油机的故障诊断。首先建立神经网络诊断模型，然后收集某型号柴油机的征兆/样本集，采用一个单隐层的 RBF 网络对样本进行训练。最后，通过测试网络，验证该网络对于故障模式的识别准确率，并对故障严重程度进行定量预测。

8.6.1 问题描述

船用柴油机是一个非常复杂的机电系统，采用整机诊断模型非常复杂，如果采用一个神经网络模型将使得网络结构非常庞大，学习训练过程比较繁琐。根据故障的层次性特点，可将柴油机整机分为废气涡轮增压系统、气缸活塞组件、燃烧系统、润滑系统和燃油系统等子系统。任一子系统的故障都可能使柴油机的性能退化甚至丧失。因此，柴油机的故障诊断可以分为两个层次进行，首先由整机性能参数的退化状态判断故障的可能位置，即故障可能处于哪一个子系统，这称为第一层次诊断；然后对该子系统进一步诊断故障原因、位置和严重程度，这称为第二层次诊断。由于第一层次的诊断过程比较简单和直接，因此



不必采用神经网络模型，只有在进行第二层次诊断时才采用 RBF 神经网络。

经过对柴油机的大量故障资料进行分析，我们知道涡轮增压系统、汽缸活塞组件与燃烧系统运行中发生故障的概率最高，本节仅研究涡轮增压系统的故障诊断问题。

8.6.2 涡轮增压系统的故障诊断

1. 征兆/故障样本集的收集与设计

根据对船用低速增压柴油机的工作过程的理论分析和实际运行经验，可以确定涡轮增压系统各部件可能出现故障的原因和部位，作为故障变量即输出变量；同时确定用于区别各种故障的征兆变量作为网络输入变量。

（1）输出变量

输出变量即故障变量，包括正常工况（无故障）F_1、增压器效率下降（压气机、透平或机械效率下降等）F_2、空冷器传热恶化 F_3、透平保护格栅阻塞 F_4、透平通流部分阻塞 F_5、空气滤清器阻塞 F_6、空冷器空气侧流阻增大 F_7 和废热锅炉流阻增大 F_8。其中，F_1 到 F_8 的取值范围为[0，1]，0 表示无此故障，1 表示该故障严重。其中网络的输出变量包括 F_1 到 F_5，而 F_6、F_7 和 F_8 直接由部件特性参数诊断输出。

（2）输入变量

输入变量即征兆变量，包括排气总管温度 T_r、扫气箱压力 P_s、各缸平均燃烧最大爆发压力 P_{max}、增压器转速 n_{tc}、扫排气道压损系数、压气机出口温度 T_c、扫气箱温度 T_s、滤网压损系数、空冷器压损系数和废气锅炉压损系数，再加上柴油机负荷的一个参数，一共 11 个输入变量。它们有的是直接测量得到的参数，有的是由测量参数导出的参数，例如某个部件前后的压降是直接测量的，但采用其导出的压损系数，能更好地反应出堵塞故障的特性。

根据内燃机原理和船用柴油机技术规范的要求，可以得到柴油机的工作参数标称值，即无故障时的数据：

- 气缸排气温度：30K（绝对温度）；
- 扫气箱压力：0.06MPa；
- 最大爆发压力：1MPa；
- 增压器转速：25s^{-1}（1500 r/min）；
- 滤网压损系数：0.1；
- 空冷器压损系数：0.1；
- 废热锅炉压损系数：0.1；
- 扫排气道压损系数：0.06；
- 压气机出口温度：30K；
- 扫气箱温度 40K。

如果上述工作参数上下偏差超过以上数据、则认为柴油机有故障。其中，排气总管温度 T_r、扫气箱压力 P_s、各缸平均燃烧最大爆发压力 P_{max}、增压器转速 n_{tc}、扫排气道压损系数、压气机出口温度 T_c 和扫气箱温度 T_s 作为网络输入变量；而滤网压损系数、空冷器压损

系数和废气锅炉压损系数则作为部件特性参数，直接进行诊断。

（3）征兆/故障样本集的设计

征兆/故障样本集的正确确定是神经网络准确进行故障诊断的关键环节。涡轮增压系统的一种故障对应一个样本。为了进一步诊断故障的严重程度，这里对每个故障取两个样本，目标值分别为 0.5 和 1。为了反映机组运行负荷范围的征兆和故障的对应关系，对额定负荷（100% MCR）、部分负荷（90% MCR，75% MCR）和半负荷（50% MCR）等 4 种情况给出样本。经过统计，这部分的样本为 36 个。

由于船舶可能会远洋航行，因此把大气环境温度分为 3 部分，即 283K～294K、295K~306K 和 307K~318K，并分别以 288K、300K 和 312K 为样本中心，得到相应的样本集（样本总数为 108 个），用于训练 RBF 神经网络，从而实现柴油机运行故障的诊断。

基于 100%MCR、90%MCR、75%MCR 和 50%MCR 的发动机负荷，我们给出了大气温度为 300K、312K 和 288K 的输入样本向量和相应的输出目标向量的征兆/故障样本集。限于篇幅，这里只列出了温度为 288K 时的 36 组征兆/故障样本数据中的 9 组数据，并利用这部分数据进行网络训练，如表 8-11 所示。

表 8-11 征兆/故障样本数据（环境温度：288K）

样本序号	输入向量	目标向量
1	0 0 0 0 0 0 0 0.500	0 0 0 0 0
2	0.935 –0.576 –0.892 –0.900 0.088 –0.604 –0.069 0.499	0 1 0 0 0
3	0.253 –0.301 –0.304 –0.431 0.031 –0.298 –0.038 0.499	0 0.5 0 0 0
4	0.277 0.166 0.106 0.200 –0.030 0.179 0.390 0.499	0 0 1 0 0
5	0.085 0.058 0.039 0.070 –0.010 0.061 0.122 0.499	0 0 0.5 0 0
6	0.657 -0.319 –0.351 –0.548 0.118 –0.419 –0.057 0.499	0 0 0 1 0
7	0.256 –0.166 –0.179 –0.289 0.065 –0.225 –0.034 0.499	0 0 0 0.5 0
8	0.615 0.698 0.252 0.687 0.092 0.685 0.065 0.499	0 0 0 0 1
9	0.128 0.255 0.239 0.254 0.043 0.245 0.020 0.499	0 0 0 0 0.5

2. 网络输出向量设计

从便于诊断的角度出发，将故障的严重程度分为 3 级，即 1 级故障、2 级故障和无故障（正常工况）。根据网络的输出向量，结果处理如下：

若 $0.75<F_i<1.50$，则 $F_i=1$ 级（严重故障）；

若 $0.25<F_i<0.75$，则 $F_i=2$ 级（中等故障）；

若 $F_i<0.25$ 或者 $F_i>1.50$，则 $F_i=$正常（无故障）。

由此可判定故障的严重程度。

8.6.3 网络设计

径向基（RBF）网络主要包括隐含层和输出层，其中隐含层的传递函数为 radbas，输出层的传递函数为纯线性函数 purelin。如图 8-7 所示，径向基网络的隐含层有 S^1 个神经元，输出层有 S^2 个神经元。

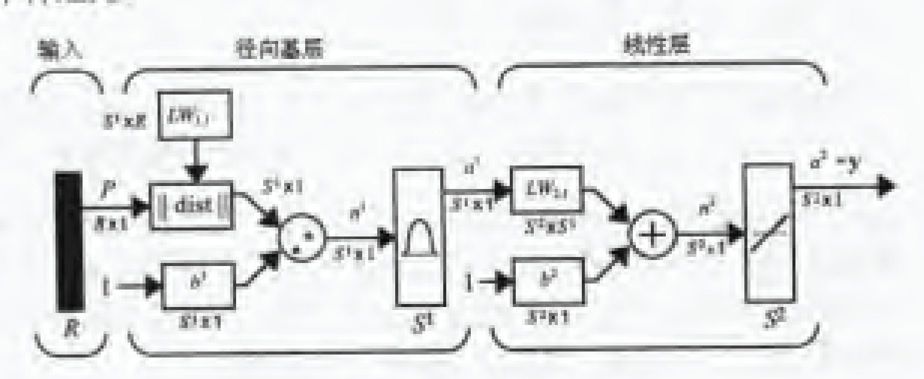


图 8-7 径向基网络结构

神经网络工具箱中用于创建 RBF 网络的函数为 newrbe，在设计过程中，最重要的参数是径向基函数的分布常数。由于本例中的样本数目不是很大，我们将分布常数设定为 1.2。由此，可利用如下 MATLAB 代码创建一个 RBF 网络：

```
spread=1.2;
net=newrbe(P,T,spread);
```

其中 P 和 T 分别对应输入向量和目标向量，它们可从表 8-11 中得到。

```
P=[0 0 0 0 0 0 0 0;
   0.935 -0.576 -0.892 -0.900 0.088 -0.604 -0.069 0.5;
   0.253 -0.301 -0.304 -0.431 0.031 -0.298 -0.038 0.499;
   0.277 0.166 0.106 0.200 -0.030 0.179 0.390 0.499;
   0.085 0.058 0.039 0.070 -0.010 0.061 0.122 0.499;
   0.657 -0.319 -0.351 -0.548 0.118 -0.419 -0.057 0.499;
   0.256 -0.166 -0.179 -0.289 0.065 -0.225 -0.034 0.499;
   0.615 0.698 0.252 0.687 0.092 0.685 0.065 0.499;
   0.128 0.255 0.239 0.254 0.043 0.245 0.020 0.499];
T=[0  0  0  0  0;
   0  1  0  0  0;
   0 0.5  0  0  0;
   0  0  1  0  0;
   0  0  0.5 0  0;
   0  0  0  1  0;
   0  0  0  0.5 0;
   0  0  0  0  1;
   0  0  0  0  0.5];
P=P';
T=T';
```



当网络的输入变量确定后，需要进行归一化处理，这里将其变换在[－1,1]的范围内，经过归一化处理的数据对于神经网络更容易训练和学习。因为原始数据幅值大小不一，有时候相差还比较悬殊。如果直接投入使用，测量值大的波动就垄断了神经网络的学习过程，使其不能反映小的测量值的变化。归一化后的数据是通过计算测量值与相应的无故障情况下基准值的偏差值，再除以最大偏差值的绝对值来获得输入变量的。



在利用函数 newrbe 创建 RBF 网络过程中，可以自动增加隐含层的神经元个数，直到均方误差满足要求为止。所以，网络的创建过程就是训练过程了。

接下来通过一组实际数据对网络进行测试，看网络是否可以正确诊断出涡轮增压系统中的故障。如果可以正确诊断出故障，则网络可以投入使用。分别在 F_2、F_3 和 F_4 有故障时进行测量，得到各参数的值，将这些数据作为输入向量，利用仿真函数计算网络输出，通过故障判别准则看是否出了故障。测试数据如表 8-12 所示。

表 8-12　测试数据

数据序号	输入向量	故障原因
1	0.920 -0.526 -0.885 -0.970 0.090 -0.611 -0.071 0.502	F_2有严重故障
2	0.091 0.061 0.036 0.068 -0.012 0.059 0.131 0.499	F_3有中等故障
3	0.671 -0.315 -0.362 -0.552 0.121 -0.417 -0.061 0.499	F_4有严重故障

测试代码如下：

```
%p1，p2，p3 分别对应上表中序号为 1，2，3 的数据
p1=[0.920 -0.526 -0.885 -0.970 0.090 -0.611 -0.071 0.502]';
p2=[0.091 0.061 0.036 0.068 -0.012 0.059 0.131 0.499]';
p3=[0.671 -0.315 -0.362 -0.552 0.121 -0.417 -0.061 0.499]';
%对网络进行仿真
y1=sim(net,p1)
y2=sim(net,p2)
y3=sim(net,p3)
```

测试结果如表 8-13 所示。

表 8-13　测试结果

输出结果	故障类别
0　0.9955　-0.0704　0.0021　0.0069	F2 有严重故障
0　-0.0086　0.0082　1.0219　-0.0007	F3 有中等故障
0　-0.0005　0.5318　0.0042　-0.0073	F4 有严重故障

分析结果后发现，网络成功地诊断出了所有故障。因此，可以将该网络投入实际工程应用了。



本书限于篇幅，采用的样本量比较少，这样一来，故障诊断的数据范围就不会很大，对于那些和训练数据相差很大的数据，网络可能无法做出正确的诊

断。在这种情况下，如果想提高网络的故障识别准确率，建议采用大容量的训练样本。

本实例完整的 MATLAB 代码如下：

```
P=[0 0 0 0 0 0 0 0.5;
   0.935 -0.576 -0.892 -0.900 0.088 -0.604 -0.069 0.499;
   0.253 -0.301 -0.304 -0.431 0.031 -0.298 -0.038 0.499;
   0.277 0.166 0.106 0.200 -0.030 0.179 0.390 0.499;
   0.085 0.058 0.039 0.070 -0.010 0.061 0.122 0.499;
   0.657 -0.319 -0.351 -0.548 0.118 -0.419 -0.057 0.499;
   0.256 -0.166 -0.179 -0.289 0.065 -0.225 -0.034 0.499;
   0.615 0.698 0.252 0.687 0.092 0.685 0.065 0.499;
   0.128 0.255 0.239 0.254 0.043 0.245 0.020 0.499];
T=[0   0   0   0   0;
   0   1   0   0   0;
   0 0.5   0   0   0;
   0   0   1   0   0;
   0   0   0.5 0   0;
   0   0   0   1   0;
   0   0   0   0.5 0;
   0   0   0   0   1;
   0   0   0   0   0.5];
P=P';
T=T';
spread=1.2;
net=newrbe(P,T,spread);

p1=[0.920 -0.526 -0.885 -0.970 0.090 -0.611 -0.071 0.502]';
p2=[0.091 0.061 0.036 0.068 -0.012 0.059 0.131 0.499]';
p3=[0.671 -0.315 -0.362 -0.552 0.121 -0.417 -0.061 0.499]';
y1=sim(net,p1)
y2=sim(net,p2)
y3=sim(net,p3)
```

8.7 小　　结

本章研究了利用神经网络工具箱来实现基于神经网络故障诊断的 5 个实例，通过这些研究，使得神经网络工具箱从理论上升到了实际应用。综合分析以上诊断实例，我们可以发现：

（1）基于神经网络的故障诊断方法在系统参数未知的情况下能够自动建立动态模型，对于线性系统和非线性系统都有很好的跟踪能力，因此可以准确地识别出系统存在的故障。

（2）通过改进或者选用高效的训练学习算法，可以在同等的收敛要求下，获得较高的诊断精度。

（3）特征提取是基于神经网络进行故障诊断的关键问题，所提取的特征需要具有代表性和典型性，能够表征系统的故障状态。另外，特征的形式要便于进行数学处理，以作为网络的训练样本和测试样本。

（4）样本的数量和质量能够在很大程度上影响神经网络的诊断精度。所谓数量，指的是训练样本的数目要充足，对于同一种故障状态，要尽量多地提取相应的特征数据；所谓质量，指的是样本数据中需要蕴含所有可能的故障状态特征数据。这样既可以提高网络的推理能力，又可提高泛化能力。

基于神经网络的故障诊断已经逐渐走向成熟，神经网络工具箱的应用可以在很大程度上解决神经网络训练和仿真过程中繁琐的数学计算问题，从而提高诊断的效率和水平，提高故障诊断的信息化程度。

第 9 章　基于神经网络的预测

在系统建模、辨识和预报中，对于线性系统，在频域，传递函数矩阵可以很好地表达系统的黑箱式输入/输出模型；在时域，利用自回归滑动平均（ARMA）模型通过各种参数估计方法，也可以给出系统输入/输出的描述。这样一来，线性系统预测问题就已经比较完美的解决了。但对于非线性系统，一般采用基于非线性自回归滑动平均（NARMA）模型进行预测，但是，很难为这种模型找到一个恰当的参数估计方法。因此，可以说传统的非线性系统辨识，在理论研究和实际应用方面，都存在极大的困难。

相比之下，神经网络在这方面显示出了明显的优越性。由于神经网络具有通过学习逼近任意非线性映射的能力，将神经网络应用于非线性系统的建模与辨识，可以不受非线性模型的限制，便于给出工程上易于实现的学习算法。

本章以几个实例为背景，介绍了如何利用神经网络工具箱，实现基于神经网络的预测。主要内容为：

- 基于神经网络的预测原理
- 电力系统负荷预报的 MATLAB 实现
- 河道浅滩演变预测的 MATLAB 实现
- 地震预报的 MATLAB 实现
- 交通运输能力预测的 MATLAB 实现
- 股市预测的 MATLAB 实现
- 财务失败预测的 MATLAB 实现
- 农作物虫情预测的 MATLAB 实现

9.1 引　　言

目前在系统建模与预报中，应用最多的是静态的多层前向神经网络，这主要是因为这种网络具有逼近任意非线性映射的能力。利用静态的多层前向神经网络建立系统的输入/输出模型，本质上就是基于网络逼近能力，通过学习获知系统差分方程中的位置非线性函数。对于静态系统的建模预报，多层前向网络能够取得良好的效果。但在实际应用中，需要建模和预报的多为非线性动态系统，利用多层前向神经网络必须事先给定模型的阶次，即预先确定系统的模型类，这一点非常难做到。

近来，具有内部反馈的动态网络在系统建模与预报中的应用已经得到重视，代表了神经网络建模与预报的发展方向。主要是因为动态网络本身就是动态时变系统，对于动力学系统建模有着自然的反映系统动态变化的能力，不需预先确定系统的模型类和阶次。但是，目前有关基于动态网络的建模和预报的研究，`还不如静态多层前向网络研究得那么深入，还缺乏特别令人信服的动态网络模型，动态网络的逼近能力还没有严谨的理论成果，学习

算法还需要进一步的完善。

尽管仍然存在一些关键性的理论问题尚未解决，但许多研究成果表明，神经网络在非线性系统的预报方面有着广泛的应用前景。下一节将介绍如何利用神经网络进行预测。

9.2 基于神经网络的预测原理

9.2.1 正向建模

正向建模是指训练一个神经网络表达系统正向动态的过程，这一过程建立的神经网络模型称为正向模型。正向模型的结构如图 9-1 所示，其中神经网络与待辨识的系统并联，两者的输出误差用做网络的训练信号。显然，这是一个典型的有教师学习问题，实际系统作为教师，向神经网络提供算法所需的期望输出。当系统是被控对象或传统控制器时，神经网络多采用多层前向网络的形式，可直接选用 BP 网络或它的各种变形。而当系统为性能评价器时，则可选择再励学习算法，这时网络既可以采用具有全局逼近能力的网络，如多层感知器，也可选用具有局部逼近能力的网络，如小脑模型关节控制器（CMAC）等。

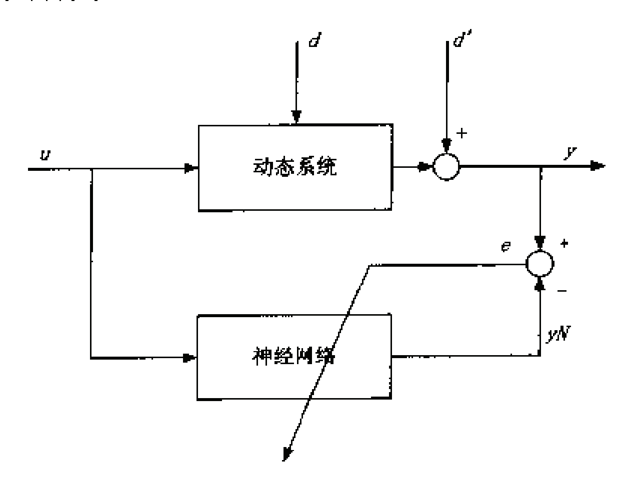


图 9-1 正向建模结构

9.2.2 逆向建模

建立动态系统的逆模型，在神经网络控制中起着关键作用，并且得到了特别广泛的应用。下面介绍其中比较简单的直接逆建模法。

直接逆建模也称为广义逆学习，如图 9-2 所示。从原理上说，这是一种最简单的方法。由图可见，拟预报的系统输出作为网络的输入，网络输出与系统输入比较，相应的输入误差用于训练，因此网络将通过学习建立系统的逆模型。但是如果所辨识的非线性系统是不可逆的，利用上述方法，将得到一个不正确的逆模型。因此，在建立系统逆模型时，可逆性应该事先有所保证。

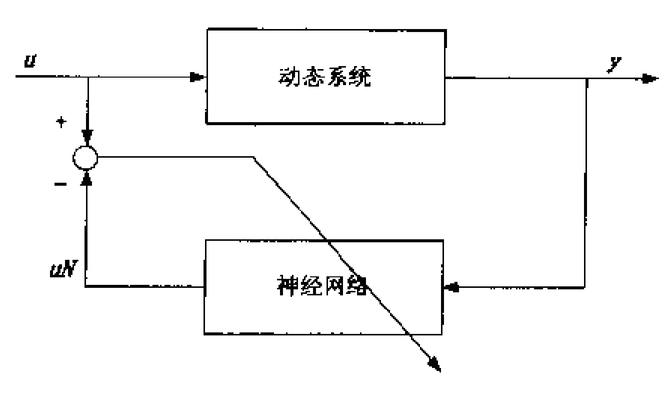


图 9-2　直接逆建模

为了获得良好的逆动力学特性，应妥善选择网络训练所需的样本集，使其比未知系统的实际运行范围更大。但实际工作时的输入信号很难事先给定，因为控制目标是使系统输出具有期望的运动，对于未知被控系统期望输入不可能给出。另一方面，在系统预报中，为保证参数估计算法的一致收敛，必须使用一定的持续激励的输入信号。对于神经网络，这是一个仍有待于进一步研究的问题。

下一节将研究如何利用神经网络来实现电力系统的负荷预测。

9.3　电力系统负荷预报的 MATLAB 实现

电力负荷预报在实时控制和保证电力系统经济、安全和可靠运行方面起着重要作用，它已经成为电力系统中现代能量管理系统的一个主要组成部分，尤其是短期负荷预报对于系统运行和生产费用具有非常重大的影响。负荷预报的误差将导致运行和生产费用的剧增，因此，精确的预报对于电力部门和供电系统都有着重要的经济意义。因此，如何提高短期预报的精度就成了电力工作者和其他科技人员致力解决的问题。

负荷预测对电力系统控制、运行和计划都有着重要意义。电力系统负荷变化受多方面影响，一方面，负荷变化存在着由未知不确定因素引起的随机波动；另一方面，又具有周期变化的规律性，这也使得负荷曲线具有相似性。同时，由于受天气、节假日等特殊情况影响，又使负荷变化出现差异。由于神经网络所具有的较强的非线性映射等特性，它常被用于负荷预测。

9.3.1　问题描述

电力系统负荷短期预报问题的解决办法和方式可以分为统计技术、专家系统法和神经网络法等 3 种。统计技术中所用的短期负荷模型一般可归为时间系列模型和回归模型。时间系列模型的缺点在于不能充分利用对负荷性能有很大影响的气候信息和其他因素，导致了预报的不准确和数据的不稳定。回归模型虽然考虑了气象信息等因素，但需要事先知道负荷与气象变量之间的函数关系，这是比较困难的。而且为了获得比较精确的预报结果，需要大量的计算，这一方法不能处理气候变量和与负荷之间的非平衡暂态关系。专家系统法利用了专家的经验知识和推理规则，使节假日或有重大活动日子的负荷预报精度得到了



提高。但是，把专家知识和经验等准确地转化为一系列规则是非常不容易的。

众所周知，负荷曲线是与很多因素相关的一个非线性函数。对于抽取和逼近这种非线性函数，神经网络是一种合适的方法。神经网络的优点在于它具有模拟多变量而不需要对输入变量做复杂的相关假定的能力。它不依靠专家经验，只利用观察到的数据，可以从训练过程中通过学习来抽取和逼近隐含的输入/输出非线性关系。近年来的研究表明，相对于前两种方法，利用神经网络技术进行电力系统短期负荷预报可获得更高的精度。

在对短期负荷进行预报前，一个特别重要的问题是如何划分负荷类型或日期类型。纵观已经发表的文献资料，大体有以下几种划分模式：

（1）将一周的 7 天分为工作日（星期一到星期五）和休息日（星期六和星期天）等两种类型；

（2）将一周分为星期一、星期二到星期四、星期五、星期六和星期日等 5 种类型。

（3）将一周的 7 天每天都看做一种类型，共有 7 种类型。

本例采用第 3 种负荷划分模式，把每一天不加区分地看做不同的类型。

9.3.2 输入/输出向量设计

在预测日的前一天中，每隔 2 个小时对电力负荷进行 1 次测量，这样一来，一天共测得 12 组负荷数据。由于负荷值曲线相邻的点之间不会发生突变，因此后一时刻的值必然和前一时刻的值有关，除非出现重大事故等特殊情况。所以这里将前一天的实时负荷数据作为网络的样本数据。

此外，由于电力负荷还与环境因素有关，比如最高和最低气温等。因此，还需要通过天气预报等手段获得预测日的最高气温、最低气温和天气特征值（晴天、阴天还是雨天）。因此形式表示天气特征值：0 表示晴天，0.5 表示阴天和 1 表示雨天。这里将电力负荷预测日当天的气象特征数据作为网络的输入变量。因此，输入变量就是一个 15 维的向量。

显而易见，目标向量就是预测日当天的 12 个负荷值，即一天中每个整点的电力负荷。这样一来，输出变量就成为一个 12 维的向量。

获得输入和输出变量后，要对其进行归一化处理，将数据处理为区间[0,1]之间的数据。归一化方法有很多种形式，这里采用如下公式：

$$\hat{x}=\frac{x-x_{\min}}{x_{\max}-x_{\min}}$$

近几年，我国南方一直处于“电荒”的被动境况，为了更好地利用电能，必须做好电力负荷的短期预报工作。这里以南方某缺电城市的 2004 年 7 月 10 日到 7 月 20 日的整点有功负荷值，以及 2004 年 7 月 11 日到 7 月 21 日的气象特征状态量作为网络的训练样本，预测 7 月 21 日的电力负荷，如表 9-1 所示，其中所有的数据都已经归一化了。

在样本中，输入向量为预测日前天的电力实际负荷数据，目标向量是预测日当天的电力负荷。由于这都是实际的测量值，因此，这些数据可以对网络进行有效的训练。如果从提高网络精度的角度出发，一方面可以增加网络训练样本的数目，另一方面还可以增加输入向量的维数。即，或者增加每天的测量点，或者把预测日前几日的负荷数据作为输入向

量。目前，训练样本数目的确定没有通用的方法，一般认为，样本过少可能使得网络的表达不够充分，从而导致网络外推的能力不够；而样本过多可能会出现样本冗余现象，既增加了网络的训练负担，也有可能出现信息量过剩使得网络出现过拟合现象。总之，样本的选取过程需要注意代表性、均衡性和用电负荷的自身特点，从而选择合理的训练样本。

表 9-1　用电负荷及气象特征

样本日期	电力负荷	气象特征
2004-7-10	0.2452 0.1466 0.1314 0.2243 0.5523 0.6642 0.7015 0.6981 0.6821 0.6945 0.7549 0.8215	
2004-7-11	0.2217 0.1581 0.1408 0.2304 0.5134 0.5312 0.6819 0.7125 0.7265 0.6847 0.7826 0.8325	0.2415 0.3027 0
2004-7-12	0.2525 0.1627 0.1507 0.2406 0.5502 0.5636 0.7051 0.7352 0.7459 0.7015 0.8064 0.8156	0.2385 0.3125 0
2004-7-13	0.2016 0.1105 0.1243 0.1978 0.5021 0.5232 0.6819 0.6952 0.7015 0.6825 0.7825 0.7895	0.2216 0.2701 1
2004-7-14	0.2115 0.1201 0.1312 0.2019 0.5532 0.5736 0.7029 0.7032 0.7189 0.7019 0.7965 0.8025	0.2352 0.2506 0.5
2004-7-15	0.2335 0.1322 0.1534 0.2214 0.5623 0.5827 0.7198 0.7276 0.7359 0.7506 0.8092 0.8221	0.2542 0.3125 0
2004-7-16	0.2368 0.1432 0.1653 0.2205 0.5823 0.5971 0.7136 0.7129 0.7263 0.7153 0.8091 0.8217	0.2601 0.3198 0
2004-7-17	0.2342 0.1368 0.1602 0.2131 0.5726 0.5822 0.7101 0.7098 0.7127 0.7121 0.7995 0.8126	0.2579 0.3099 0
2004-7-18	0.2113 0.1212 0.1305 0.1819 0.4952 0.5312 0.6886 0.6898 0.6999 0.7323 0.7721 0.7956	0.2301 0.2867 0.5
2004-7-19	0.2005 0.1121 0.1207 0.1605 0.4556 0.5022 0.6553 0.6673 0.6798 0.7023 0.7521 0.7756	0.2234 0.2799 1
2004-7-20	0.2123 0.1257 0.1343 0.2079 0.5579 0.5716 0.7059 0.7145 0.7205 0.7401 0.8019 0.8136	0.2314 0.2977 0
2004-7-21	0.2119 0.1215 0.1621 0.2161 0.6171 0.6159 0.7155 0.7201 0.7243 0.7298 0.8179 0.8229	0.2317 0.2936 0

9.3.3　BP 网络设计

BP 网络是系统预测中应用特别广泛的一种网络形式，因此，这里采用 BP 网络对负荷值进行预报。根据 BP 网络的设计网络，一般的预测问题都可以通过单隐层的 BP 网络实现。由于输入向量有 15 个元素，所以网络输入层的神经元有 15 个，根据 Kolmogorov 定理，可知网络中间层的神经元可以取 31 个。而输出向量有 12 个，所以输出层中的神经元应该有 12 个。网络中间层的神经元传递函数采用 S 型正切函数 tansig，输出层神经元传递函数采用 S 型对数函数 logsig。这是因为函数的输出位于区间[0,1]中，正好满足网络输出的要求。

利用以下代码创建一个满足上述要求的 BP 网络：

```
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[31,12],{'tansig','logsig'},'trainlm')
```

其中，变量 threshold 用于规定输入向量的最大值和最小值，规定了网络输入向量的最大值为 1，最小值为 0。'trainlm'表示设定网络的训练函数为 trainlm，它采用 Levenberg-Marquardt 算法进行网络学习。

9.3.4 网络训练

网络经过训练后才可以用于电力负荷预测的实际应用。考虑到网络的结构比较复杂，神经元个数比较多，需要适当增大训练次数和学习速率。训练参数的设定如表 9-2 所示。

表 9-2 训练参数

训练次数	训练目标	学习速率
1000	0.01	0.1

训练代码如下：

```
net.trainParam.epochs=1000;
net.trainParam.goal=0.01;
LP.lr=0.1;
net=train(net,P,T);  %P 为输入向量，T 为目标向量
```

训练结果为：

```
TRAINLM, Epoch 0/1000, MSE 0.244221/0.01, Gradient 7.25261/1e-010
TRAINLM, Epoch 5/1000, MSE 0.00873744/0.01, Gradient 0.930479/1e-010
TRAINLM, Performance goal met.
```

可见，经过 5 次训练后，网络误差达到要求，结果如图 9-3 所示。

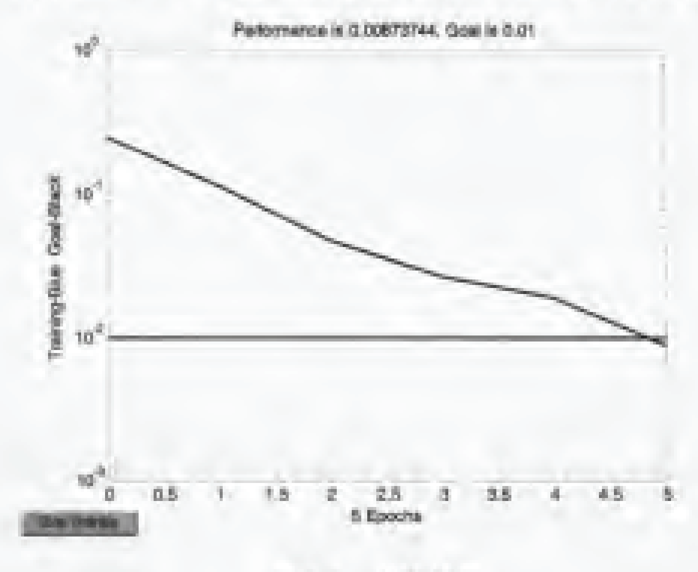


图 9-3 训练结果

训练好的网络还需要进行测试才可以判定是否可以投入实际应用。这里的测试数据就是利用表 9-1 中的 2004 年 7 月 20 日的电力负荷和 21 日的气象特征数据来预测 21 日的用

电负荷来检验预测误差能否满足要求的。代码如下：

```
%P_test 为由 20 日的电力负荷向量和 21 日的气象特征组成的测试向量
Out=sim(net,P_test);
```

这里利用仿真函数 sim 来计算网络的输出，运行结果为：

```
Out=
    0.2120 0.1209 0.1609 0.2150 0.6178 0.6161 0.7231 1.0000 0.7238 0.7277 0.8236 0.8234
```

预报误差曲线如图 9-4 所示，由图可见，网络预测值和真实值之间的误差是非常小的，除了第 8 次出现了一个相对比较大的误差之外，其余的误差都在 0 左右。即使是第 8 次的误差也只有 0.28，这完全满足应用要求。

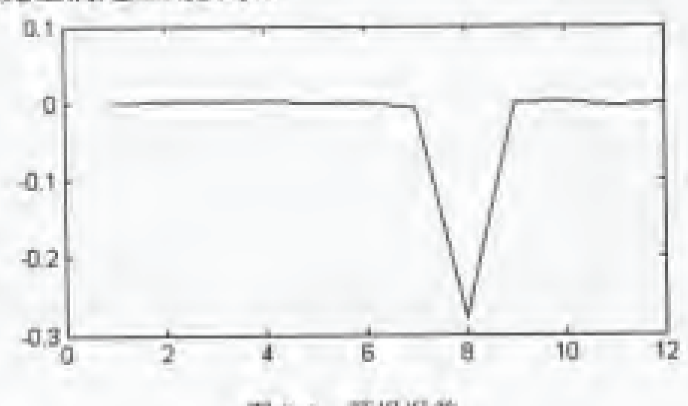


图 9-4　预报误差

本例的完整 MATLAB 代码如下：

```
P=[0.2452 0.1466 0.1314 0.2243 0.5523 0.6642 0.7015 0.6981 0.6821 0.6945 0.7549 0.8215 0.2415
  0.3027 0;
  0.2217 0.1581 0.1408 0.2304 0.5134 0.5312 0.6819 0.7125 0.7265 0.6847 0.7826 0.8325 0.2385
  0.3125 0;
  0.2525 0.1627 0.1507 0.2406 0.5502 0.5636 0.7051 0.7352 0.7459 0.7015 0.8064 0.8156 0.2216
  0.2701 1;
  0.2016 0.1105 0.1243 0.1978 0.5021 0.5232 0.6819 0.6952 0.7015 0.6825 0.7825 0.7895 0.2352
  0.2506 0.5;
  0.2115 0.1201 0.1312 0.2019 0.5532 0.5736 0.7029 0.7032 0.7189 0.7019 0.7965 0.8025 0.2542
  0.3125 0;
  0.2335 0.1322 0.1534 0.2214 0.5623 0.5827 0.7198 0.7276 0.7359 0.7506 0.8092 0.8221 0.2601
  0.3198 0;
  0.2368 0.1432 0.1653 0.2205 0.5823 0.5971 0.7136 0.7129 0.7263 0.7153 0.8091 0.8217 0.2579
  0.3099 0;
  0.2342 0.1368 0.1602 0.2131 0.5726 0.5822 0.7101 0.7098 0.7127 0.7121 0.7995 0.8126 0.2301
  0.2867 0.5;
  0.2113 0.1212 0.1305 0.1819 0.4952 0.5312 0.6886 0.6898 0.6999 0.7323 0.7721 0.7956 0.2234
  0.2799 1;
  0.2005 0.1121 0.1207 0.1605 0.4556 0.5022 0.6553 0.6673 0.6798 0.7023 0.7521 0.7756 0.2314
  0.2977 0]';
T=[0.2217 0.1581 0.1408 0.2304 0.5134 0.5312 0.6819 0.7125 0.7265 0.6847 0.7826 0.8325;
   0.2525 0.1627 0.1507 0.2406 0.5502 0.5636 0.7051 0.7352 0.7459 0.7015 0.8064 0.8156;
   0.2016 0.1105 0.1243 0.1978 0.5021 0.5232 0.6819 0.6952 0.7015 0.6825 0.7825 0.7895;
   0.2115 0.1201 0.1312 0.2019 0.5532 0.5736 0.7029 0.7032 0.7189 0.7019 0.7965 0.8025;
```

```
    0.2335 0.1322 0.1534 0.2214 0.5623 0.5827 0.7198 0.7276 0.7359 0.7506 0.8092 0.8221;
    0.2368 0.1432 0.1653 0.2205 0.5823 0.5971 0.7136 0.7129 0.7263 0.7153 0.8091 0.8217;
    0.2342 0.1368 0.1602 0.2131 0.5726 0.5822 0.7101 0.7098 0.7127 0.7121 0.7995 0.8126;
    0.2113 0.1212 0.1305 0.1819 0.4952 0.5312 0.6886 0.6898 0.6999 0.7323 0.7721 0.7956;
    0.2005 0.1121 0.1207 0.1605 0.4556 0.5022 0.6553 0.6673 0.6798 0.7023 0.7521 0.7756;
    0.2123 0.1257 0.1343 0.2079 0.5579 0.5716 0.7059 0.7145 0.7205 0.7401 0.8019 0.8136]';
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[31,12],{'tansig','logsig'},'trainlm');
net.trainParam.epochs=1000;
net.trainParam.goal=0.01;
LP.lr=0.1;
net=train(net,P,T);

P_test=[0.2123 0.1257 0.1343 0.2079 0.5579 0.5716 0.7059 0.7145 0.7205 0.7401 0.8019 0.8136
        0.2317 0.2936 0]';
Out=sim(net,P_test);
%绘制预报误差曲线
X=[0.2119 0.1215 0.1621 0.2161 0.6171 0.6159 0.7155 0.7201 0.7243 0.7298 0.8179 0.8229];
plot(1:12,X-Out)
```

9.4 河道浅滩演变预测的MATLAB实现

河道中的浅滩碍航、滞洪，因此，无论水利还是水运部门都对浅滩的演变规律和浅滩的治理非常重视。浅滩演变是河床演变的一部分，侧重枯水河床变化分析研究。浅滩演变预测的任务是根据过去和现在浅滩演变的情况，预测研究河段浅滩断面年内最小水深和浅滩年平均淤积厚度。目前对浅滩演变预测的泥沙数学模型的研究尽管取得了很大发展，但由于水、沙和边界的复杂性，很难考虑预测中的各种随机因素；物理模型在试验时对水、沙模拟也存在一定的困难，且研究费用高；实测资料分析法，以定性分析和推理为主，所得结论受研究者的经验影响。因此，有必要根据浅滩演变的特点探索浅滩演变预测的新方法。浅滩河段可视为一个非线性动态系统，浅滩的演变实质是系统对输入的水流、泥沙进行调整的结果。神经网络模型具有很强的非线性映射能力和柔性网络结构，以及高度的容错性和鲁棒性，适于非线性问题的研究。因此，神经网络模型的特点正好适合于浅滩演变的预测。

9.4.1 基于BP网络的演变预测

BP网络由于结构简单，具有较强非线性映射能力，是应用最为广泛的一类神经网络。因此，这里首先采用BP网络建立预测模型。

1. 影响因子分析

从通航目的出发，河道浅滩演变预测关注的是浅滩处年平均淤积厚度和浅滩断面年内最小水深。因此，预测模型的输出因子是浅滩断面年内最小水深 H_{min} 和年平均淤积厚度 Δ

Z。

输入因子的选取与浅滩的形成和影响浅滩形成变化的诸因素有关。从河床演变理论及河流地貌动力学角度出发，可以判断，影响浅滩断面年内最小水深和浅滩年平均淤积厚度的主要因素有：

- 上游来流量 Q 和来流过程 $Q \sim t$；
- 上游输沙量 G 和输沙过程 $Q \sim t$，泥沙组成 $d_s \sim P_s$；
- 河段比降 J；
- 河床形态 $\sqrt{B}/H$ 和床沙组成 $d_p \sim P_b$；
- 人类活动的影响。

在上述因素中，前 4 个因素对不同的河流影响的程度是不相同的。对冲积性平原河流，第 1、第 2 两个因素起主导作用。对于山区河流，第 3 和第 4 两个因素常起主要作用。第 5 个因素对河段冲淤的影响，视不同情况，其影响程度和范围不同。

在上述各因素中，河段平均比降难以获取，但是，比降的变化与研究河段的上下游水位密切相关，而上游水位与上游流量通常具有相关性，人工神经网络要求各输入因子相互独立，因此上游水位不计入输入因子，下游水位 Z_D 视为一个输入因子。床沙组成与推移质输沙率密切相关，因为获得长期床沙组成逐年变化的资料不容易，而选推移质输沙率作为输入因子。河道浅滩演变预测的人工神经网络模型的输入因子有：$Q, Q \sim t, G, G \sim t, g_b, d_s \sim P_s, \sqrt{B}/H$ 和 Z_D。于是，浅滩断面年内最小水深 $H_{\min}$ 和年平均淤积厚度 ΔZ 可表示为：

$$H_{\min} = f_1(Q, Q \sim t, G, G \sim t, d_s \sim P_s, g_b, \sqrt{B}/H, Z_D)$$
$$\Delta Z = f_2(Q, Q \sim t, G, G \sim t, d_s \sim P_s, g_b, \sqrt{B}/H, Z_D)$$

对于具体的研究河段，视问题的具体情况，还可以对上述因子做进一步的取舍。

2. 样本中各因子的获取

在浅滩断面年内最小水深和年平均淤积厚度预测模型建立中，输入和输出因子的获取是关键环节。一般而言，系统输入因子中：$Q, Q \sim t, G, G \sim t, d_s \sim P_s$ 和 Z_D 为水文资料，可从研究河段上下游水文站获取。$g_b, \sqrt{B}/H$ 和 $H_{\min}$,利用地理信息系统技术和图像处理技术，通过研究河段多年的地形图，并进行数字化处理，得到数字化冲淤等值图和栅格地图，选取每年的浅滩部位进行统计来获取。

由于是选取年平均情况作为样本因子进行预测计算，因此所需资料时间系列较长，常常会出现具有代表性的某一年份的地形资料缺测或遗失。这时可采用水深遥感技术，对缺损地形图的年份进行插补。具体步骤如下。

（1）选取与缺测年份相邻的前后各一个年份的同期或相近时期的资料。

（2）用缺测年份相邻的前一年和后一年的实测地形资料确定水深遥感模式，并分别反演出两个时相的遥感水深。

（3）对缺测的那一年，采用“借地率定”方法率定水深遥感模式中的系数。方法一：选取与卫片资料相近时期，浅滩河段相邻水域有实测资料的断面（如水文站点的大断面）进行模式率定；方法二：选取与浅滩河段相邻或本河段内地形基本稳定、变化很小的部分

水域进行模式率定。

（4）用率定后的水深遥感模式反演缺测的那一年的水下地形。

（5）用相邻的3个年份遥感水深计算两两之间的冲淤值，并在地理信息系统中产生冲淤等值图，供统计浅滩上有关地理信息的使用。

3. 实例分析

这里以某河段为研究对象，该河段河床由中粗沙组成，岸滩多粘土和沙土，抗冲性差，易崩塌。从泥沙来源、河床多年的演变情况，以及从泥沙数学模拟和物理模型试验的结果看，河段河床演变以推移形式为主。河段进口有A地水文站，出口受B地水位站控制，水文资料获取方便。此外，本河段还有较多年份的地形资料，为河床边界条件提供了资料。

对于具体的研究河段，上述两式中的影响因子还需进一步分析选取。该河段河床的冲淤变形以底沙为主，$Q\sim t$ 和 $G\sim t$ 的变化主要对悬沙冲淤变化显著的河段影响较大，而对底沙冲淤变化为主的河段影响相对较小，加之预测的是年平均情况，因此，$Q\sim t$ 和 $G\sim t$ 对河床冲淤的影响可忽略不计。就年变化情况而言，输入因子中的来流量取年平均流量，来沙量取年总输沙量。河段来沙组成 $d_s\sim P_s$ 在20世纪70年代与80年代基本一致，20世纪90年代以后受水口大坝蓄水发电影响，与80年代泥沙组成有所差别，但90年代与2000年的泥沙组成基本一致，预测样本所选时段为1990～2000年，在此时段内，$d_s\sim P_s$ 对不同年份河床冲淤的影响可以不计。从推移质输沙率神经网络模型试验研究看，$g_b=f$（来水条件，来沙条件，河床边界条件，输沙反馈条件，出流反馈条件），也就是说推移质输沙率不是真正意义上的独立变量，因此，g_b 也可以去掉。这样，上述两式中的影响因子就简化为 Q，G，Z_D 和 $\sqrt{B}/H$。

该河段输入因子中 Q，G 和 Z_D 从A地和B地的水文站获取。$\sqrt{B}/H$，H_{min} 和 ΔZ 从航道图和冲淤等值图中提取。样本是研究对象与网络模型的接口。正确选取训练样本是非常重要的一步，它包括样本特征选取及样本数目确定。样本特征应该能很好地反映这类问题的基本特征，不仅在训练区内有代表性，而且在预测区内有普遍性。这要求浅滩演变预测样本具有普遍性，又要考虑特殊水文现象的影响。该河段于1989年开始建水口电站，建电站以后河段的演变受电站影响，考虑到这一人为因素影响，研究时段选1990～2000年。在这11年里，该河段水文动力因素变化大，既有电站蓄水影响，又包含了各种水文条件的变化情况，还包含有来水、来沙各种组合情况，样本具有一致性。此外，在这一时段中，1992年为丰水丰沙年，1996年为小水少沙年，1998年为该河段所在流域特大洪水中沙年，也考虑了特殊性。因此，该时段的样本覆盖性能比较好，基本上涵盖了研究河段的各种水文情况，预测样本从水文条件的变化看既有代表性也有普遍性。根据样本数据选取原则，选择1990~1999年的数据作为网络训练样本，如表9-3所示。

表9-3 样本数据

年 份	Q	G	Z_D	$\sqrt{B}/H$	H_{min}	ΔZ
1990	1520	510	5.155	33.88	0.7	1.9
1991	1468	521	5.321	35.79	0.6	1.798
1992	2412	1140	5.32	25.89	0.8	1.289

（续表）

年　份	Q	G	Z_D	$\sqrt{B}/H$	$H_{\min}$	ΔZ
1993	1750	129	4.7	23.8	1	1.68
1994	1688	361	4.865	27.08	0.8	1.149
1995	1607	489	5.1	28.9	1.03	1.72
1996	1200	127	4.56	19.84	1.8	1.095
1997	1990	148	4.89	29.373	0.9	1.230
1998	1509	511	5.12	34.3	0.8	1.35
1999	1730	133	4.46	23.06	1.4	1.201

获得样本数据向量后，由于其中各个指标互不相同，原始样本中各向量的数量级差别很大，为了计算方便及防止部分神经元达到过饱和状态，在研究中对样本的输入进行归一化处理。

可以利用 MATLAB 实现向量的归一化过程，这里将样本数据都归一化到区间[0,1]之间。令 P 表示输入向量，T 表示目标向量，归一化代码为：

```
%p 为原始输入数据
p=[
   1520     510  5.155 33.88;1468 521    5.321 35.79;
   2412     1140 5.32  25.89;1750 129    4.7    23.8;
   1688     361  4.865 27.08;1607 489    5.1    28.9;
   1200     127  4.56  19.84;1990 148    4.89   29.373;
   1509     511  5.12  34.3;1730  133    4.46   23.06]';
   %t 为原始目标数据
t=[0.7       1.9;0.6      1.798;0.8 1.289;1 1.68;0.8 1.149;
   1.03 1.72;1.8 1.095;0.9     1.230;0.8 1.35;1.4 1.201]';
%P,T 分别表示归一化后的输入向量和目标向量
for i=1:4
     P(i,:)=(p(i,:)-min(p(i,:)))/(max(p(i,:))-min(p(i,:)));
end
for i=1:2
     T(i,:)=(t(i,:)-min(t(i,:)))/(max(t(i,:))-min(t(i,:)));
end
```

接下来设计网络的拓扑结构，由于单隐层 BP 网络的非线性映射能力比较强，这里采用单隐层的神经网络，而中间层的神经元个数需要通过实验来确定。由于输入神经元有 4 个，所以中间层神经元个数应该在 9~16 之间。因此，这里为中间层神经元个数选择 3 个值，分别为 9、12 和 15，并分别检查网络性能。

不同个数的隐单元组成的 BP 网络训练曲线分别如图 9-5、9-6 和 9-7 所示。通过比较发现，中间层神经元个数为 9 和 12 时，网络的收敛速度比较快。

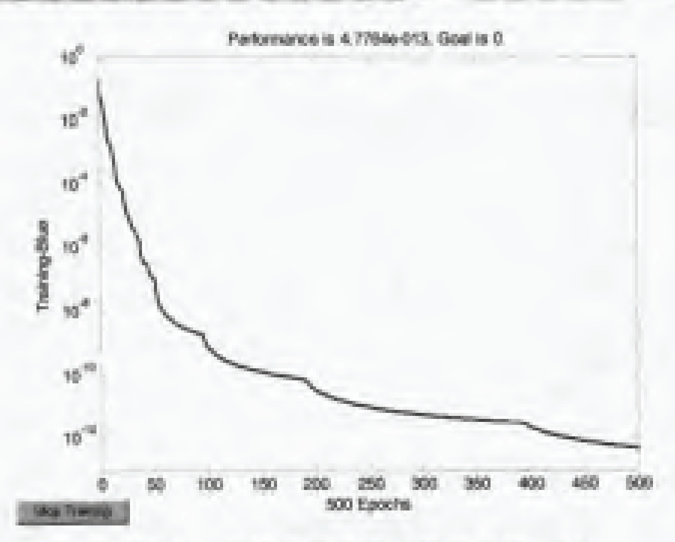


图 9-5　训练误差曲线（隐单元数：9）

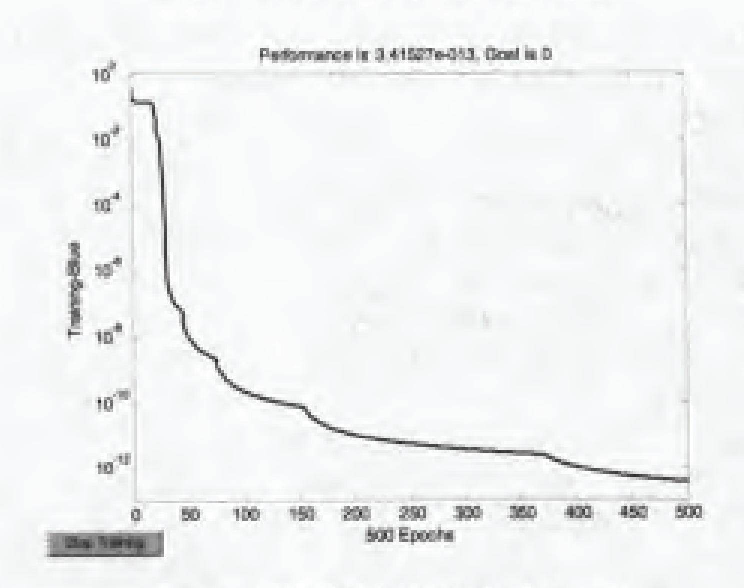


图 9-6　训练误差曲线（隐单元数：12）

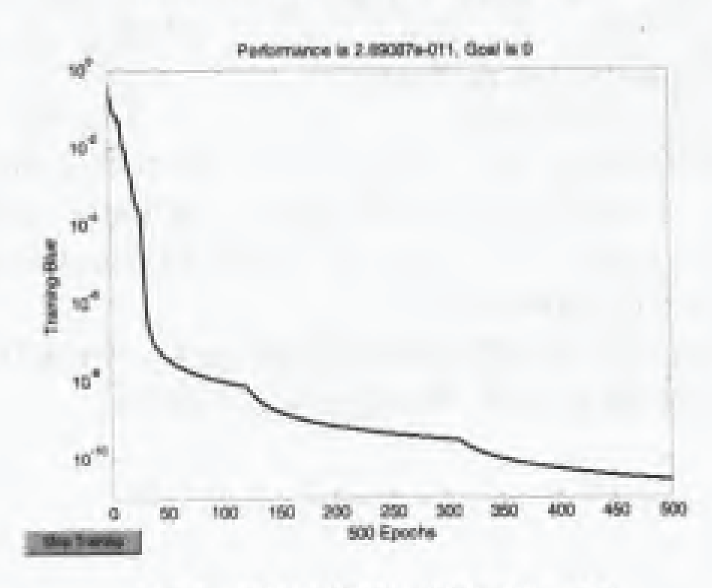


图 9-7　训练误差曲线（隐单元数：15）

接下来观察网络的预测性能，在表 9-3 中的样本数据中选取 1990 年、1992 年、1994 年、1996 年、1997 年和 1999 年的数据作为网络的测试数据。预测误差曲线如图 9-8 所示。

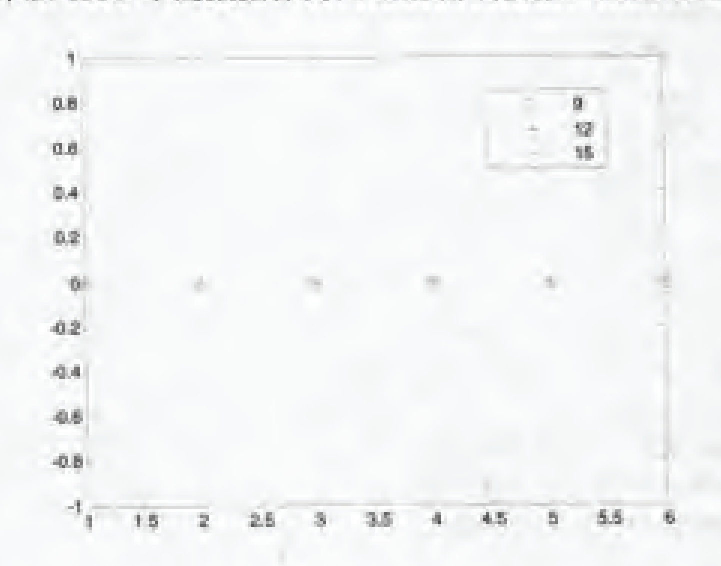


图 9-8 预测误差曲线

由图 9-8 可见，不同结构的 BP 网络的预测误差都为 0，这是因为测试样本是从训练样本中选取的。由于训练的精度非常高，网络对每一组数据都精确地拟合了，所以出现预测误差为 0 的情况。要更准确地测试网络的性能，必须从训练样本以外的数据样本中选取测试样本。

本实例的完整 MATLAB 代码为：

```
p=[
  1520    510  5.155 33.88;1468 521   5.321 35.79;
  2412    1140 5.32  25.89;1750 129   4.7   23.8;
  1688    361  4.865 27.08;1607 489   5.1   28.9;
  1200    127  4.56  19.84;1990 148   4.89  29.373;
  1509    511  5.12  34.3;1730  133   4.46  23.06]';
t=[0.7    1.9;0.6    1.798;0.8 1.289;1 1.68;0.8 1.149;
  1.03 1.72;1.8 1.095;0.9    1.230;0.8 1.35;1.4 1.201]';
%归一化后的输入向量 P
for i=1:4
    P(i,:)=(p(i,:)-min(p(i,:)))/(max(p(i,:))-min(p(i,:)));
end
%归一化后的目标向量 A
for i=1:2
    A(i,:)=(t(i,:)-min(t(i,:)))/(max(t(i,:))-min(t(i,:)));
end
%测试样本
P_test=[P(:,1) P(:,3) P(:,5) P(:,7) P(:,8) P(:,10)];
T_test=[A(:,1) A(:,3) A(:,5) A(:,7) A(:,8) A(:,10)];
%隐层单元个数向量
No=[9 12 15];
for i=1:3
```

```
        net=newff(minmax(P),[No(i),2],{'tansig','logsig'});
        net.trainParam.epochs=500;
        net=init(net);
        net=train(net,P,A);
        Temp=sim(net,P_test);
        y(2*i-1,:)=Temp(1,:);
        y(2*i,:)=Temp(2,:)
    end
    Y1=[y(1,:);y(2,:)];
    Y2=[y(3,:);y(4,:)];
    Y3=[y(5,:);y(6,:)];
    %求预测误差
    for i=1:6
        error1(i)=norm(Y1(:,i)-T_test(:,i));
        error2(i)=norm(Y2(:,i)-T_test(:,i));
        error3(i)=norm(Y3(:,i)-T_test(:,i));
    end
    figure;
    plot(1:6,error1);
    hold on;
    plot(1:6,error2,'--');
    hold on;
    plot(1:6,error3,'+');
    hold off;
```

9.4.2 基于 RBF 网络的演变预测

上节介绍了如何利用 BP 网络进行演变预测，可以看出，BP 网络预测精度比较高，但训练误差收敛比较慢。本节基于同样的背景，尝试利用 RBF 网络进行演变预测。

1. 网络创建

由于输入向量 P 和目标向量 T 都已经获得，因此，创建网络之前惟一需要确定的就是径向基函数的分布密度 SPREAD。由于 SPREAD 的选择对网络性能有着比较重要的影响，因此，这里通过选择多个不同的 SPERAD 来确定最佳值。MATLAB 代码如下：

```
    for i=1:5
        net=newrbe(P,T,i);
        temp=sim(net,P)
        y(2*i-1,:)=temp(1,:);
        y(2*i,:)=temp(2,:);
    end
```

上述代码一共创建了 5 个 RBF 网络，径向基函数的分布密度分别为 1,2,3,4,5。输出结果为：

```
    y=
      Columns 1 through 8
      0.0833    -0.0000     0.1667     0.3333     0.1667     0.3583     1.0000     0.2500
```

```
1.0000    0.8733    0.2410    0.7267    0.0671    0.7764    0.0000    0.1677
0.0833   -0.0000    0.1667    0.3333    0.1667    0.3583    1.0000    0.2500
1.0000    0.8733    0.2410    0.7267    0.0671    0.7764    0.0000    0.1677
0.0833    0.0000    0.1667    0.3333    0.1667    0.3583    1.0000    0.2500
1.0000    0.8733    0.2410    0.7267    0.0671    0.7764    0.0000    0.1677
0.0833    0.0000    0.1667    0.3333    0.1667    0.3583    1.0000    0.2500
1.0000    0.8733    0.2410    0.7267    0.0671    0.7764    0.0000    0.1677
0.0833   -0.0000    0.1667    0.3333    0.1667    0.3583    1.0000    0.2500
1.0000    0.8733    0.2410    0.7267    0.0671    0.7764    0.0000    0.1677
Columns 9 through 10
  0.1667    0.6667
  0.3168    0.1317
  0.1667    0.6667
  0.3168    0.1317
  0.1667    0.6667
  0.3168    0.1317
  0.1667    0.6667
  0.3168    0.1317
  0.1667    0.6667
  0.3168    0.1317
```

所有网络对结果的预测都是一致的，这是因为测试数据就等于训练数据，而函数 newrbe 建立了一个准确的 RBF 网络，为了更加准确地测试网络的预测性能，需要从其他水文资料寻找与训练样本不一致的样本数据。

9.4.3 结论

神经网络作为一种输入/输出的高度非线性映射，通过对作用函数的多次复合，实现了浅滩演变预测中输入与输出之间的高度非线性映射。它具有以下的优点：

（1）不需要建立浅滩演变的数学方程。

（2）具有自适应性和学习功能。

（3）模型预测的规律受样本的性质影响，只要样本选择得当，模型预测精度就由网络结构确定。

本例使用的 BP 网络和 RBF 网络模型，经过对某河段的演变预测验算，其结果与实测计算值吻合理想。实验表明，利用 BP 神经网络和 RBF 神经网络模型对浅滩变化进行预测是可行的。

9.5 地震预报的 MATLAB 实现

地震预报是地理问题研究领域中一个重要课题，准确的地震预报可以帮助人们及时采取有效措施，降低人员伤亡和经济损失。引发地震的相关性因素很多，其产生机理的复杂性、孕育过程的非线性和认识问题的困难性使得人们很难建立较完善的物理理论模型。对有关物理参数加以精确的描述，只能借助一些观测到的相关现象进行分析、总结和推理。

相对于传统的预报方法，神经网络在处理这方面问题中有着独特的优势，主要体现在：

（1）容错能力强。由于网络的知识信息采用分布式存储，个别单元的损坏不会引起输出错误。这就使得预测或识别过程中容错能力强，可靠性高。

（2）预测或识别速度快。训练好的网络在对未知样本进行预测或识别时仅需要少量的加法和乘法，使得其运算速度明显快于其他方法。

（3）避开了特征因素与判别目标的复杂关系描述，特别是公式的表述。网络可以自己学习和记忆各输入量和输出量之间的关系。

本节将根据某地的地震资料，研究如何利用神经网络工具箱，实现基于神经网络的地震预报。

9.5.1 概述

以我国西南某地震常发地区的地震资料作为样本来源，实现基于神经网络的地震预报。根据这些地震资料，提取出 7 个预报因子和实际发生的震级 M 作为输入和目标向量。预报因子为：

- 半年内 $M\geq3$ 的地震累计频度；
- 半年内能量释放积累值；
- b 值；
- 异常地震群个数；
- 地震条带个数；
- 是否处于活动期内；
- 相关地震区地震震级。

一共收集了 10 个学习样本，如表 9-4 所示。

表 9-4 学习震例

地震累计频度	累计释放能量	b 值	异常地震群个数	地震条带个数	活动周期	相关区震级	实际震级
0	0	0.62	0	0	0	0	0
0.3915	0.4741	0.77	0.5	0.5	1	0.3158	0.5313
0.2835	0.5402	0.68	0	0.5	1	0.3158	0.5938
0.6210	1.0000	0.63	1	0.5	1	1.0000	0.9375
0.4185	0.4183	0.67	0.5	0	1	0.7368	0.4375
0.2160	0.4948	0.71	0	0	1	0.2632	0.5000
0.9990	0.0383	0.75	0.5	1	1	0.9474	1.0000
0.5805	0.4925	0.71	0	0	0	0.3684	0.3750
0.0810	0.0692	0.76	0	0	0	0.0526	0.3125
0.3915	0.1230	0.98	0.5	0	0	0.8974	0.6563

表 9-4 中的前 7 项为学习样本中的输入因子，输出因子为实际震级，利用上表中的学习震例对网络进行训练。在训练前，应该对数据进行归一化处理，表 9-4 中数据已经是归一化后的数据了。

9.5.2 BP网络设计

还是采用单隐层的BP网络进行地震预测。由于输入样本为7维的输入向量，因此，输入层一共有7个神经元，则中间层应该有15个神经元、网络只有1个输出数据，则输出层只有1个神经元。因此，网络应该为7×15×1的结构。按照BP网络的一般设计原则，中间层神经元的传递函数为S型正切函数。由于输出已被归一化到区间[0,1]中，因此，输出层神经元的传递函数可以设定为S型对数函数。利用如下代码创建一个符合上述要求的BP网络：

```
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[15,1],{'tansig','logsig'},'traingdx');
```

其中，threshold设定了网络输入向量的取值范围[0,1]，网络所用的训练函数为traingdx，该函数以梯度下降法进行学习，并且学习速率是自适应的。

中间层的神经元个数是很难确定的，而这又在很大程度上影响着网络的预测性能。这里首先取15个，然后，观察网络性能；之后，再分别取10和20，并与此时的预测性能进行比较，检验中间层神经元个数对网络性能的影响。当网络的预测误差最小时，网络中间层的神经元数目就是最佳值。

9.5.3 BP网络训练与测试

对于上一小节得到的网络，利用表9-4中的数据进行训练。训练后的网络才有可能满足实际应用的要求。训练参数的设定如表9-5所示，其他参数取默认值。

表9-5 训练参数

训练次数	训练目标
1000	0.01

网络训练的代码如下：

```
net.trainParam.epochs =1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
```

变量P和T分别表示网络的输入向量和目标向量，它们是从表9-4中得出的。训练结果为：

```
TRAINGDX, Epoch 0/1000, MSE 0.0811768/0.001, Gradient 0.17954/1e-006
TRAINGDX, Epoch 25/1000, MSE 0.0705269/0.001, Gradient 0.159156/1e-006
TRAINGDX, Epoch 50/1000, MSE 0.0494224/0.001, Gradient 0.102915/1e-006
TRAINGDX, Epoch 75/1000, MSE 0.0303765/0.001, Gradient 0.041892/1e-006
TRAINGDX, Epoch 100/1000, MSE 0.0151122/0.001, Gradient 0.0254611/1e-006
TRAINGDX, Epoch 125/1000, MSE 0.00539131/0.001, Gradient 0.0052669/1e-006
TRAINGDX, Epoch 150/1000, MSE 0.0037484/0.001, Gradient 0.00237267/1e-006
TRAINGDX, Epoch 175/1000, MSE 0.00214641/0.001, Gradient 0.00146846/1e-006
TRAINGDX, Epoch 200/1000, MSE 0.00162598/0.001, Gradient 0.0111882/1e-006
TRAINGDX, Epoch 225/1000, MSE 0.00147946/0.001, Gradient 0.00134714/1e-006
```

```
TRAINGDX, Epoch 250/1000, MSE 0.00131055/0.001, Gradient 0.00115809/1e-006
TRAINGDX, Epoch 270/1000, MSE 0.000998859/0.001, Gradient 0.00091321/1e-006
TRAINGDX, Performance goal met
```

可见经过 270 次训练后，网络的目标误差达到要求，如图 9-9 所示。

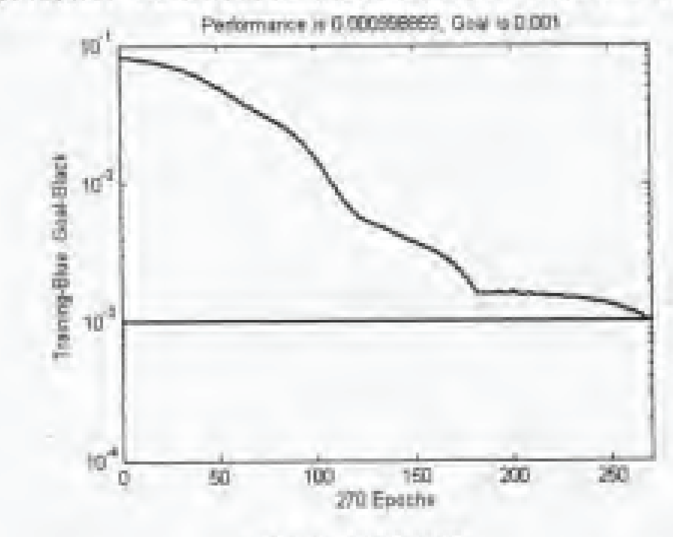


图 9-9　训练结果

网络训练结束后，还必须利用另外一组地震数据对其进行测试，数据如表 9-6 所示。所谓测试，实际上是利用仿真函数来获得网络的输出，然后检查输出和实际测量值之间的误差是否满足要求。

测试代码为：

```
P_test=[0.0270  0.0742   0.62 0    0    0    0.2105;
        0.1755  0.3667   0.77 0    0.5  1    0.7368;
        0.4320  0.3790   0.68 0.5  0    1    0.2632;
        0.4995  0.4347   0.63 0    0    1    0.6842;
        0.6885  0.5842   0.67 0.5  0.5  1    0.4211;
        0.5400  0.8038   0.71 0.5  0.5  1    0.5789;
        0.1620  0.2565   0.75 0    0    1    0.4737]';
Y=sim(net,P_test)
```

输出：Y=

```
        0.2360   0.3865   0.3571   0.5433   0.6591   0.6807   0.3518
```

表 9-6　测试数据

地震累计频度	累计释放能量	b 值	异常地震群个数	地震条带个数	活动周期	相关区震级	实际震级
0.0270	0.0742	0.62	0	0	0	0.2105	0.1875
0.1755	0.3667	0.77	0	0.5	1	0.7368	0.4062
0.4320	0.3790	0.68	0.5	0	1	0.2632	0.4375
0.4995	0.4347	0.63	0	0	1	0.6842	0.5938
0.6885	0.5842	0.67	0.5	0.5	1	0.4211	0.6250
0.5400	0.8038	0.71	0.5	0.5	1	0.5789	0.7187
0.1620	0.2565	0.75	0	0	1	0.4737	0.3750

输出结果经过反归一化处理后得到预报震级，和实际震级相比较可得到网络的预报误

差，如表9-7所示。

表9-7 预报误差

实际震级	预报震级	预报误差
4.4	4.5550	0.1550
5.1	5.0369	0.0631
5.2	4.9626	0.2374
5.7	5.5387	0.1613
5.8	5.9393	0.1393
6.1	5.9782	0.0218
5.0	4.9258	0.0742

由表9-7可见，网络的预报误差比较小，因此，性能可以满足实际应用的要求。预报误差曲线如图9-10所示。



图9-10 网络的预报误差

接下来将网络中间层数目设置为10和20，并分别进行训练和仿真，得到如下一组训练误差曲线和预测误差曲线。其中，如图9-11所示，为中间层神经元个数为10的情况下的训练误差曲线；如图9-12所示，为中间层神经元个数为20的情况下的训练误差曲线；如图9-13所示，为两者误差对比曲线。

此时针对测试数据得到的仿真结果为：

```
    Y =
        0.1058    0.4123    0.3637    0.4578    0.7514    0.7596    0.3171
```

可见，这种情况下网络的预报误差比较大。

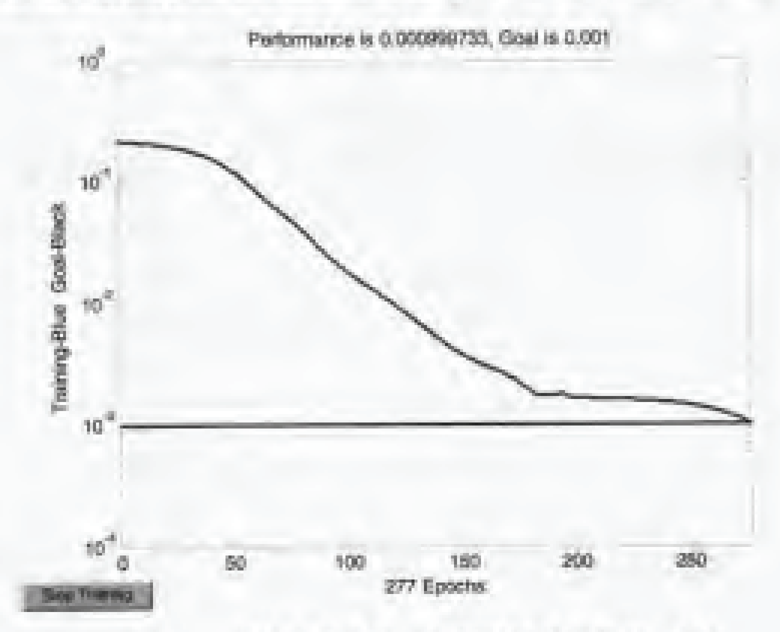


图 9-11　训练误差曲线（中间层神经元数目：10）

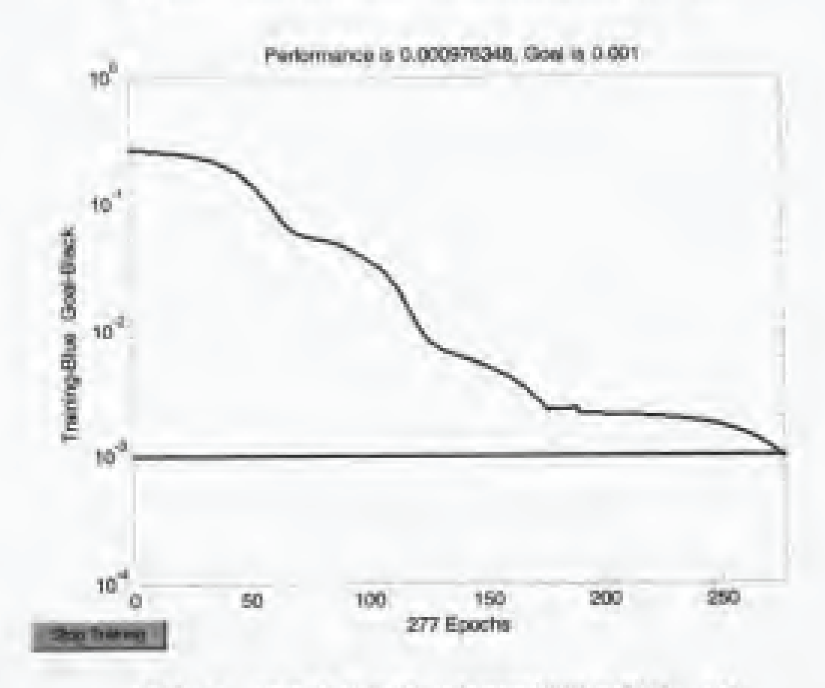


图 9-12　训练误差曲线（中间层神经元数目：20）

此时网络的仿真结果为：

```
Y =
     0.2611    0.8092    0.3097    0.5392    0.7457    0.8754    0.6545
```

可见误差非常大，将 3 种情况下的误差进行对比，如图 9-13 所示，可见中间层神经元个数为 15 时，网络的预测性能最好。

这也证明了一个重要的结论，就是中间层神经元个数的增加，虽然可以提高网络的映射精度，但并不意味着一定就会提高网络的性能。因此，我们在设计 BP 网络时，不能无限制地增加中间层神经元的个数。

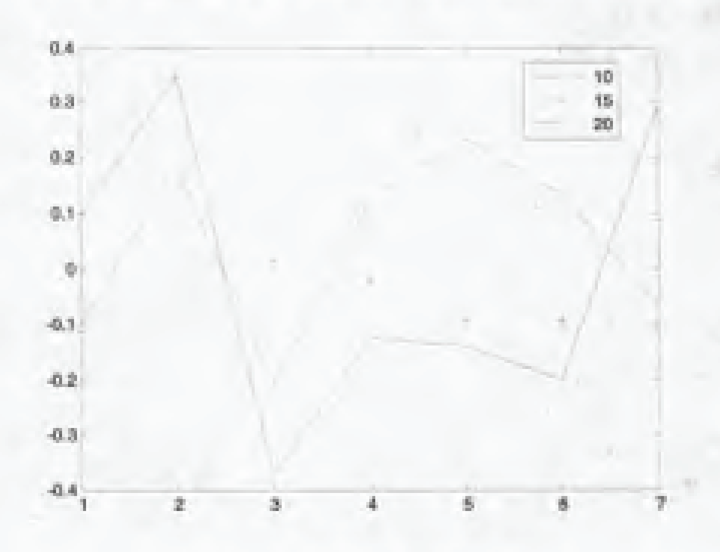


图9-13 预测误差对比曲线

完整的MATLAB代码为：

```
%输入向量P和目标向量T
P=[0 0      0.62  0     0     0;
   0.3915  0.4741    0.77  0.5   0.5   1     0.3158;
   0.2835  0.5402    0.68  0     0.5   1     0.3158;
   0.6210  1.0000    0.63  1     0.5   1     1.0000;
   0.4185  0.4183    0.67  0.5   0     1     0.7368;
   0.2160  0.4948    0.71  0     0     1     0.2632;
   0.9990  0.0383    0.75  0.5   1     1     0.9474
   0.5805  0.4925    0.71  0     0     0     0.3684;
   0.0810  0.0692    0.76  0     0     0     0.0526;
   0.3915  0.1230    0.98  0.5   0     0     0.8974]';
T=[0 0.5313 0.5938 0.9375 0.4375 0.5000 1.0000 0.3750 0.3125 0.6563];
%测试向量P_test和实际输出T_test
P_test=[0.0270  0.0742    0.62  0     0     0     0.2105;
        0.1755  0.3667    0.77  0     0.5   1     0.7368;
        0.4320  0.3790    0.68  0.5   0     1     0.2632;
        0.4995  0.4347    0.63  0     0     1     0.6842;
        0.6885  0.5842    0.67  0.5   0.5   1     0.4211;
        0.5400  0.8038    0.71  0.5   0.5   1     0.5789;
        0.1620  0.2565    0.75  0     0     1     0.4737]';
T_test=[0.1875 0.4062 0.4375 0.5938 0.6250 0.7187 0.3750];
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1];
a=[10 15 20];
for i=1:3
    net=newff(threshold,[a(i),1],{'tansig','logsig'},'traingdx');
    net.trainParam.epochs =1000;
    net.trainParam.goal=0.001;
%init 函数用于将网络初始化
    net=init(net);
```

```
        net=train(net,P,T);
        Y(i,:)=sim(net,P_test);
    end
    figure;
    %绘制误差曲线
    %中间层神经元个数为 10
    plot(1:7,Y(1,:)-T_test);
    hold on;
    %中间层神经元个数为 15
    plot(1:7,Y(2,:)-T_test,'+');
    hold on;
    %中间层神经元个数为 20
    plot(1:7,Y(3,:)-T_test,'--');
    hold off;
```

9.5.4 地震预测的竞争网络模型

自组织竞争神经网络能够对输入模式进行自组织训练和判断，并将其最终分为不同的类型。与 BP 神经网络方法相比，这种自组织、自适应的学习能力进一步拓宽了人工神经网络在模式识别、分类方面的应用。在地震预报中，有时需要根据各种地震活动性指标（包括前兆指标）将发生在不同时间、空间和强度的地震进行归类研究，根据这些样本的特征对其他样本进行外推预报。分类方法有模糊聚类、投影寻踪和神经网络等。这里采用自组织竞争网络对某地的震例进行分类研究。

利用自组织竞争网络进行地震预报，首先应该提取有关地震预报的重要指标，确定网络结构。这里以我国某地及其邻近地区从 1978 年到 1989 年的地震趋势作为检验实例，研究的时间为 1 年，所选的 11 项地震活动指标为：

- 次数最多的地震震级；
- b 值；
- 平均震级；
- 平均纬度；
- 平均纬度偏差；
- 平均经度；
- 平均经度偏差；
- 最大地震震级；
- ML 大于 115 的地震次数；
- 相邻两年的地震次数差，当绝对值超过均值一个数量级时，先除以 10 再取整；
- 相邻两年最大地震的震级差，当其为负数时作乘 0.1 处理。

所有的实际地震数据如表 9-8 所示。利用前 10 年的数据参加竞争训练，最后 1 年的数据作为测试样本。按照震级的大小分为：一般地震、中等地震和严重地震 3 类，因此这里需要设置神经元数目为 3 个。为了加快学习速度，将学习速率设定为 0.1。表 9-8 的数据是已经归一化处理以后的数据。

本实例的完整 MATLAB 代码如下：

```
P=[0.3125 0.3125 0 0 0.1875 0 1.0000 0.5000 0.1875 0.5000;
   0.45 0.49 0.65 0.60 0.50 0.62 0.36 0.43 0.42 0.43;
   0.4902 0.3333 0.7647 0.0196 0.3137 0 1.0000 0.5686 0.6471 0.6078;
   0.7639 0.8611 1.0000 0.8889 0.5972 0.8194 0 0.1528 0.7917 0.6528;
   0.93 0.57 0.96 0.94 0.80 0.96 0.53 0.70 1.12 0.89;
   0.4643 0 0.1786 0.3214 0.1786 1.0000 0.1786 0.1429 0.2857 0.3214;
   0.1765 0 0.0588 0.1765 0.3529 0.2353 1.0000 0.9412 0.5882 0.6471;
   3.9 6.3 3.5 3.9 5.0 3.5 6.3 4.1 5.1 5.4;
   0.0473 0.8581 0.2162 0.1081 0.1419 0 1.0000 1.0000 0.0405 0.0405;
   0.5 12 0.03 0.2 1.0 0.2 15 0 0.02 0;
   0.1 2.4 0.28 0.4 1.1 0.15 2.8 0.22 1.0 0.30];
%创建竞争型网络，竞争层神经元个数为 3，学习速率为 0.1
net = newc(minmax(P),3,0.1);
%对网络进行训练
net=init(net);
net=train(net,P);
%测试网络
y=sim(net,P);
y=vec2ind(y)
```

表 9-8 地震活动指标年值

年份	次数最多的震级	b 值	平均震级	纬度偏差	经度偏差	平均纬度	平均经度	最大震级	次数	次数差	震级差
1978	0.3125	0.45	0.4902	0.7639	0.93	0.4643	0.1765	3.9	0.0473	0.5	0.1
1979	0.3125	0.49	0.3333	0.8611	0.57	0	0	6.3	0.8581	12	2.4
1980	0	0.65	0.7647	1.0000	0.96	0.1786	0.0588	3.5	0.2162	0.03	0.28
1981	0	0.60	0.0196	0.8889	0.94	0.3214	0.1765	3.9	0.1081	0.2	0.4
1982	0.1875	0.50	0.3137	0.5972	0.80	0.1786	0.3529	5.0	0.1419	1.0	1.1
1983	0	0.62	0	0.8194	0.96	1.0000	0.2353	3.5	0	0.2	0.15
1984	1.0000	0.36	1.0000	0	0.53	0.1786	1.0000	6.3	1.0000	15	2.8
1985	0.5000	0.43	0.5686	0.1528	0.70	0.1429	0.9412	4.1	1.0000	0	0.22
1986	0.1875	0.42	0.6471	0.7917	1.12	0.2857	0.5882	5.1	0.0405	0.02	1.0
1987	0.5000	0.43	0.6078	0.6528	0.89	0.3214	0.6471	5.4	0.0405	0	0.3
1988	0.3125	0.43	0.6078	0.8333	1.05	0.4286	0.5882	4.0	0.0878	1	0.14
1989	0.3161	0.45	0.5001	0.7853	1	0.4235	0.1825	4.1	0.0501	0.4	0.12

代码的输出结果为：

```
y=2    1    2    2    3    2    1    3    3    3
```

由此可见，表 9-8 中第 1、3、4、6 行属于一类，即 1978 年、1980 年、1981 年和 1983 年属于一类；2、7 行即 1979 年和 1984 年属于一类。其余的，1982 年、1986 年、1987 年和 1988 年属于一类。直接检查表中的数据，我们会发现属于同一类的数据是比较相似的，这也验证了以上分类的结果。

利用竞争型网络进行预报的原理为，通过训练样本对网络进行训练，训练好的网络中记忆了所有的分类模式。当输入一个新的样本后，激发了对应的神经元，就可以对新的样

本进行分类。在本例中，可以预报地震属于哪一级。

接下来利用网络预测1989年地震属于哪一级，实际上这也是对网络进行测试。通过直接进行数据对比，我们认为1989年的震级应该和1978年的一致，因为两年的数据非常接近。MATLAB代码如下：

```
P=[0.3125 0.45 0.5001 0.7853 1. 0.4235 0.1825 4.1 0.0501 0.4 0.12]';
y=sim(net,P);
y=vec2ind(y)
```

输出结果为y=3。由此可见，网络有着比较好的预报精度。

9.6 交通运输能力预测的MATLAB实现

运输系统作为社会经济系统中的一个子系统，在受外界因素影响和作用的同时，对外部经济系统也具有一定的反作用，使得运输需求同时受到来自运输系统内外两方面因素的影响。作为运输基础设施建设投资决策的基础，运输需求预测在国家和区域经济发展规划中具有十分重要的作用。其中，由于货物运输、地方经济及企业发展的紧密联系，货运需求预测成为货运需求和经济发展关系研究中的一个重要问题。因此，作为反映货物运输需求的一项重要指标，货运量预测研究和分析具有较强的实际意义。

9.6.1 背景概述

从货运量的产生来看，它是外部经济需求和运输系统供给两方面因素共同作用的结果。从外部经济系统的作用看，在经济体系内部存在许多影响货运需求的因素，将这些因素归纳起来，有两大部分：一部分属于各种经济总量因素，如国民经济发展规模、工业发展规模及基建规模等；另一部分属于各种经济结构因素，如产业结构、工业结构等。货运需求不仅受国民经济总量的影响，还要受经济结构因素的影响。从内部运输系统的作用来看，也存在类似情况。因此，货运量影响因素总体上可分为规模因素和结构因素两类，其中结构类因素主要体现在产业结构和运输结构上，产业结构中最主要的是工业结构。在国民经济发展的不同阶段，规模因素和结构因素在货运量增长中所起的作用也不同，货运量的增长变化也呈现不同的形式。同时，由于运输市场中供需非均衡性客观存在，内外部系统对货运量的影响程度不一，而且由于作用形式复杂，这就使得货运量预测具有较大的复杂性和非线性等特点。

常用的货运量预测方法包括时间序列方法（移动平滑法、指数平滑法和随机时间序列方法）、相关（回归）分析法、灰色预测方法和作为多种方法综合的组合预测方法等。这些方法大都集中在对其因果关系回归模型和时间序列模型的分析上，所建立的模型不能全面和本质地反映所预测动态数据的内在结构和复杂特性，从而丢失了信息量。人工神经网络作为一种并行的计算模型，具有传统建模方法所不具备的很多优点，有很好的非线性映射能力，对被建模对象的经验知识要求不多，一般不必事先知道有关被建模对象的结构、参数和动态特性等方面的知识，只需给出对象的输入/输出数据，通过网络本身的学习功能就可以达到输入与输出的映射关系。

货运量预测可以利用 BP 网络和 RBF 网络模型，但是这两种网络在用于预测时，存在收敛速度慢和局部极小的缺点，在解决样本量少且噪声较多的问题时，效果并不理想。广义回归神经网络 GRNN 在逼近能力、分类能力和学习速度上较 BP 网络和 RBF 网络有着较强的优势，网络最后收敛于样本量集聚较多的优化回归面，并且在样本数据缺乏时，预测效果也比较好，此外，网络还可以处理不稳定的数据。因此，本例尝试利用 GRNN 建立预测模型，对货运量进行预测。

9.6.2 网络创建与训练

GRNN（广义回归神经网络）的结构如图 3-26 所示。网络的第一层为径向基隐含层，神经元个数等于训练样本数，该层的权值函数为欧氏距离函数（用||dist||表示），其作用为计算网络输入与第一层的权值 $IW_{1,1}$ 之间的距离，b^1 为隐含层的阈值。符号“·”表示||dist||的输出与阈值 b^1 之间的关系。隐含层的传递函数为径向基函数，通常采用高斯函数作为网络的传递函数：

$$R_i(x)=\exp\left(-\frac{\|x-c\|^2}{2\sigma_i^2}\right)$$

其中，σ_i 决定了第 i 个隐含层位置中基函数的形状，σ_i 越大，则基函数越平滑，所以又称为光滑因子。

网络的第二层为线性输出层，其权函数为规范化点积权函数（用 nprod 表示），计算网络的向量 n^2，它的每个元素就是向量 a^1 和权值矩阵 $LW_{2,1}$ 每行元素的点积再除以向量 a^1 的各元素之和得到的，并将结果 n^2 提供给线性传递函数 a^2=purelin(n^2)，计算网络输出。

GRNN 连接权值的学习修正仍然使用 BP 算法，由于网络隐含层结点中的作用函数（基函数）采用高斯函数，高斯函数作为一种局部分布对中心径向对称衰减的非负非线性函数，对输入信号将在局部产生响应，即当输入信号靠近基函数的中央范围时，隐含层结点将产生较大的输出，由此看出这种网络具有局部逼近能力，这也是该网络之所以学习速度更快的原因。此外，GRNN 中人为调节的参数少，只有一个阈值，网络的学习全部依赖数据样本，这个特点决定了网络得以最大限度地避免人为主观假定对预测结果的影响。

根据对关于货运量影响因素的分析，这里分别取国内生产总值 GDP、工业总产值、铁路运输线路长度、复线里程比重、公路运输线路长度、等级公路比重、铁路货车数量和民用载货车辆数量等 8 项指标作为货运量的影响因子，以货运总量、铁轮货运量和公路货运量做出货运量的输出因子，即网络的输出。由此构建 GRNN，由于光滑因子也可以影响网络的性能，因此这里需要不断进行尝试来确定最佳值。

根据上面确定的网络输入和输出因子，利用 1995～2001 年共 7 年的历史统计数据作为网络的训练样本，2002～2003 年间共 2 年的历史统计数据作为网络的外推测试样本。输入样本和目标样本如表 9-9 所示。

表 9-9　样本数据（输入样本）

年份	GDP	工业总产值	铁路运输长度	复线比重	公路运输长度	等级公路比重	铁路货车数量	民用载货车辆
1995	58478	135185	5.46	0.23	16.5	0.21	1005.3	585.44
1996	67884	152369	5.46	0.27	18.7	0.26	1105.6	575.03
1997	74462	182563	6.01	0.25	21.6	0.28	1204.6	601.23
1998	78345	201587	6.12	0.26	25.8	0.29	1316.5	627.89
1999	82067	225689	6.21	0.26	30.5	0.31	1423.5	676.95
2000	89403	240568	6.37	0.28	34.9	0.33	1536.2	716.32
2001	95933	263856	6.38	0.28	39.8	0.36	1632.6	765.24
2002	104790	285697	6.65	0.30	42.5	0.39	1753.2	812.22
2003	116694	308765	6.65	0.30	46.7	0.41	1865.5	875.26

表 9-9（续）　样本数据（目标样本）

年　份	货　运　量	铁路货运量	公路货运量
1995	102569	52365	46251
1996	124587	60821	56245
1997	148792	69253	67362
1998	162568	79856	78165
1999	186592	91658	90548
2000	205862	99635	98758
2001	226598	109862	102564
2002	245636	120566	111257
2003	263595	130378	120356

利用以下代码将上表中的数据归一化到区间[0,1]中，需要注意的是，对于已经位于区间[0,1]的数据，无需进行归一化处理。

```
p=[58478  135185   5.46 0.23 16.5 0.21 1005.3   585.44;
   67884  152369   5.46 0.27 18.7 0.26 1105.6   575.03;
   74462  182563   6.01 0.25 21.6 0.28 1204.6   601.23;
   78345  201587   6.12 0.26 25.8 0.29 1316.5   627.89;
   82067  225689   6.21 0.26 30.5 0.31 1423.5   676.95;
   89403  240568   6.37 0.28 34.9 0.33 1536.2   716.32;
   95933  263856   6.38 0.28 39.8 0.36 1632.6   765.24;
   104790 285697   6.65 0.30 42.5 0.39 1753.2   812.22;
   116694 308765   6.65 0.30 46.7 0.41 1865.5   875.26]';
t=[102569 52365    46251;
   124587 60821    56245;
   148792 69253    67362;
   162568 79856    78165;
   186592 91658    90548;
   205862 99635    98758;
   226598 109862   102564;
   245636 120566   111257;
```

```
    263595  130378      120356]';
a=[1 2 3 5 7 8];
P=p;
for i=1:6
    P(a(i),:)=(p(a(i),:)-min(p(a(i),:)))/(max(p(a(i),:))-min(p(a(i),:)));
end
for i=1:3
    T(i,:)=(t(i,:)-min(t(i,:)))/(max(t(i,:))-min(t(i,:)));
end
```

数据处理结束后，接下来利用这些数据创建一个 GRNN 网络并进行训练与测试。由于训练样本是 1995～2001 年的数据，测试数据是 2002～2003 年的数据；此外，由于光滑因子对网络的性能影响比较大，因此，需要不断尝试才可能获得最佳值。MATLAB 代码如下：

```
%网络训练样本
%输入向量 P_train
P_train=[P(:,1) P(:,2) P(:,3) P(:,4) P(:,5) P(:,6) P(:,7)];
%目标向量 T_train
T_train=[T(:,1) T(:,2) T(:,3) T(:,4) T(:,5) T(:,6) T(:,7)];
%网络测试样本
%输入向量 P_test
P_test=[P(:,8) P(:,9)];
%目标向量 T_test
T_test=[T(:,8) T(:,9)];
for i=0.1:0.1:1
    net=newgrnn(P_train,T_train,i);
    %网络对训练数据的逼近
    y_out=sim(net,P_train)
    %网络的预测输出
    y=sim(net,P_test)
end
```

在上述代码中可以看出，将光滑因子分别设置为 0.1,0.2,···,0.5，经过对输出结果的检查发现，光滑因子越小，网络对样本的逼近性能就越强；光滑因子越大，网络对样本数据的逼近过程就越平滑。网络对训练样本的逼近误差如图 9-14 所示；网络的预测误差如图 9-15 所示。由图可见，当光滑因子为 0.1 时，无论是逼近性能还是预测性能，误差都比较小，随着光滑因子的增加，误差也在不断增长。

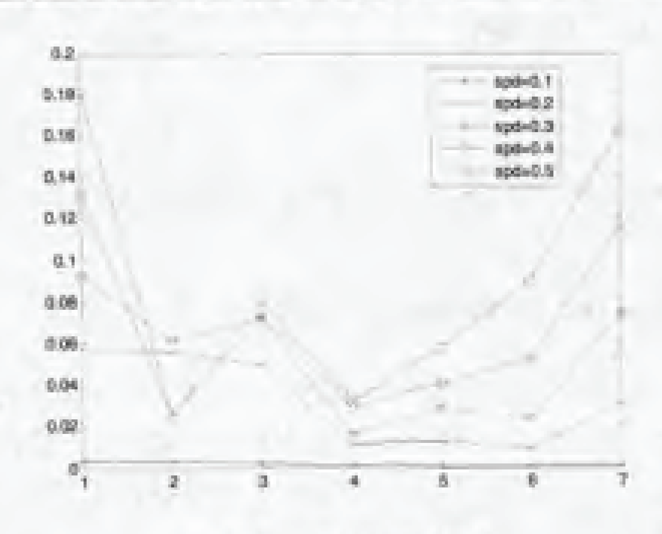


图 9-14　网络的逼近误差

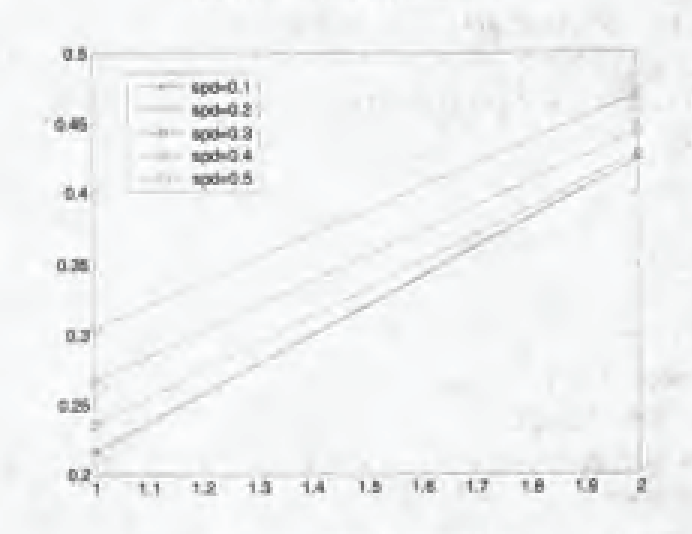


图 9-15　网络的预测误差

从误差的角度考虑，这里的光滑因子取 0.1，此时网络的测试输出为：

```
y=
        0.7702      0.7802
        0.7370      0.7570
        0.7599      0.7799
```

经过反归一化处理后，实际结果为：

```
yc=
        226590      228200
        109860      111420
        102560      104000
```

由此可见，网络的预测误差比较大，这是因为 2002～2003 年间的数据较前几年的增加幅度比较大，而且这些数据同训练数据之间的距离也比较远，外推起来有一定的难度。此外，由于训练样本容量比较小，所以预测精度不是很高。考虑到这些因素，这里的预测结果还是可以接受的。

9.6.3　结论与分析

本例在分析了货运量的影响因素和预测特点的基础上，利用 GRNN 神经网络进行了货运量预测。经过预测效果的检验和分析，证明了 GRNN 神经网络在货运量预测中应用是有效的，而且，就具体网络训练而言，与 BP 神经网络相比，由于需要调整的参数比较少，只有一个光滑因子，因此可以更快地找到合适的预测网络，具有较大的计算优势。

本实例的完整 MATLAB 代码为：

```
p=[58478  135185   5.46 0.23 16.5 0.21 1005.3    585.44;
   67884  152369   5.46 0.27 18.7 0.26 1105.6    575.03;
   74462  182563   6.01 0.25 21.6 0.28 1204.6    601.23;
   78345  201587   6.12 0.26 25.8 0.29 1316.5    627.89;
   82067  225689   6.21 0.26 30.5 0.31 1423.5    676.95;
   89403  240568   6.37 0.28 34.9 0.33 1536.2    716.32;
   95933  263856   6.38 0.28 39.8 0.36 1632.6    765.24;
   104790 285697   6.65 0.30 42.5 0.39 1753.2    812.22;
   116694 308765   6.65 0.30 46.7 0.41 1865.5    875.26]';
t=[102569 52365    46251;
   124587 60821    56245;
   148792 69253    67362;
   162568 79856    78165;
   186592 91658    90548;
   205862 99635    98758;
   226598 109862   102564;
   245636 120566   111257;
   263595 130378   120356]';
%设定需要归一化的行序号
a=[1 2 3 5 7 8];
P=p;
%对输入向量进行归一化处理
for i=1:6
    P(a(i),:)=(p(a(i),:)-min(p(a(i),:)))/(max(p(a(i),:))-min(p(a(i),:)));
end
%对目标向量进行归一化处理
for i=1:3
    T(i,:)=(t(i,:)-min(t(i,:)))/(max(t(i,:))-min(t(i,:)));
end
%训练样本，P_train 为输入向量，T_train 为目标向量
P_train=[P(:,1) P(:,2) P(:,3) P(:,4) P(:,5) P(:,6) P(:,7)];
T_train=[T(:,1) T(:,2) T(:,3) T(:,4) T(:,5) T(:,6) T(:,7)];
%测试样本，P_test 为输入向量，T_test 为目标向量
P_test=[P(:,8) P(:,9)];
T_test=[T(:,8) T(:,9)];
for i=1:5
    net=newgrnn(P_train,T_train,i/10);
    %网络对训练数据的逼近
```

```
        temp=sim(net,P_train);
        j=3*i;
        y_out(j-2,:)=temp(1,:);
        y_out(j-1,:)=temp(2,:);
        y_out(j,:)=temp(3,:);
        %网络的预测输出
        temp=sim(net,P_test);
        y(j-2,:)=temp(1,:);
        y(j-1,:)=temp(2,:);
        y(j,:)=temp(3,:);
end
y1=[y_out(1,:);y_out(2,:);y_out(3,:)];
y2=[y_out(4,:);y_out(5,:);y_out(6,:)];
y3=[y_out(7,:);y_out(8,:);y_out(9,:)];
y4=[y_out(10,:);y_out(11,:);y_out(12,:)];
y5=[y_out(13,:);y_out(14,:);y_out(15,:)];
y6=[y(1,:);y(2,:);y(3,:)];
y7=[y(4,:);y(5,:);y(6,:)];
y8=[y(7,:);y(8,:);y(9,:)];
y9=[y(10,:);y(11,:);y(12,:)];
y10=[y(13,:);y(14,:);y(15,:)];
%计算逼近误差
for i=1:7
      error1(i)=norm(y1(:,i)-T_train(:,i));
      error2(i)=norm(y2(:,i)-T_train(:,i));
      error3(i)=norm(y3(:,i)-T_train(:,i));
      error4(i)=norm(y4(:,i)-T_train(:,i));
      error5(i)=norm(y5(:,i)-T_train(:,i));
end
%计算预测误差
for i=1:2
      error6(i)=norm(y6(:,i)-T_test(:,i));
      error7(i)=norm(y7(:,i)-T_test(:,i));
      error8(i)=norm(y8(:,i)-T_test(:,i));
      error9(i)=norm(y9(:,i)-T_test(:,i));
      error10(i)=norm(y10(:,i)-T_test(:,i));
end
%绘制逼近误差曲线
plot(1:7,error1,'-*');
hold on;
plot(1:7,error2,'-+');
hold on;
plot(1:7,error3,'-h');
hold on;
plot(1:7,error4,'-d');
hold on;
```

```
plot(1:7,error5,'-o');
hold off;
figure;
%绘制预测误差曲线
plot(1:2,error6,'-*');
hold on;
plot(1:2,error7,'-+');
hold on;
plot(1:2,error8,'-h');
hold on;
plot(1:2,error9,'-d');
hold on;
plot(1:2,error10,'-o');
hold off;
```

9.7 股市预测的MATLAB实现

股票市场具有高收益和高风险并存的特性。关于股市分析和预测的研究一直被人们所重视，但是由于股票市场的高度非线性特征，导致众多股市分析方法的应用效果都难如人意。近年来，计算机技术和人工智能技术的迅猛发展，为股票市场的建模和预测提供了新的技术和方法。

神经网络因为其广泛的适应能力和学习能力，在非线性系统的预测方面得到了广泛的应用。RBF网络是一种新颖有效的前向型神经网络，由于该网络输出层是对中间层的线性加权，使得该网络避免了像BP网络那样繁琐冗长的计算，具有较高的运算速度和外推能力，同时使得网络有较强的非线性映射功能。RBF网络是通过非线性基函数的线性组合实现从输入空间 R^N 到输出空间 R^M 的非线性转换。而股票数据是一类非线性较强的时间序列，对它们进行预测，即从前 N 个数据中预测将来的 M 个数据，实质上就是找出从 R^N 到 R^M 的非线性映射关系。因此，可以说径向基网络特别适合于非线性时间序列如股票市场等系统的预测。

9.7.1 股市概述

由于股市中的数据可以看做一个时间序列进行处理，因此这里假定有时间序列 $x=\{x_i \mid x_i \in R,\quad i=1,2,\cdots,L\}$，现在希望通过序列的前 N 个时刻的值，预测出后 M 个时刻的值。这里可以采用序列的前 N 个时刻的数据为滑动窗，并将其映射为 M 个值。这 M 个值代表在该窗之后的 M 个时刻上的预测值。如表9-10所示，列出了数据的一种划分方法。该表把数据分为 K 个长度为 $N+M$ 的、有一定重叠的数据段，每一个数据段可以看做一个样本，这样就可得到 $K=L-(N+M)+1$ 个样本。这样一来，就可以将每个样本的前 N 个值作为RBF神经网络的输入，后 M 个值作为目标输出。通过学习，实现从 R^N 到输出空间 R^M 的映射，从而达到时间序列预测的目的。

表 9-10　数据的划分方法

N 个输入	M 个输出
$x_1,\cdots,x_N$	$x_{N+1},\cdots,x_{N+M}$
$x_2,\cdots,x_{N+1}$	$x_{N+2},\cdots,x_{N+M+1}$
……	……
$x_K,\cdots,x_{N+K-1}$	$x_{N+K},\cdots,x_{N+M+K-1}$

这里的样本数据来源于某公司的股票市场资料，该股票数据共有 402 个，即时间序列的长度 L=402。出于篇幅所限的原因，这里无法将所有的数据同时列出。值得指出的是，无论采用多大的学习样本，网络的设计和训练过程是一致的。惟一不同的是，通过大容量样本训练出来的网络其预报误差更小，外推能力也更强。本实例的主要目的是演示基于神经网络预测股票市场的过程，因此，这里节选了所有数据中的 10 个数据作为本例的股票数据，如表 9-11 所示。与前面的各实例一样，在学习之前首先要对这些数据进行归一化处理。表 9-11 中的数据是已经归一化后的数据。

表 9-11　股票数据

日　期	股 票 价 格
1998-1-1	0.1254
1998-1-2	0.2315
1998-1-3	0.3297
1998-1-4	0.4376
1998-1-5	0.5425
1998-1-6	0.6078
1998-1-7	0.7119
1998-1-8	0.8685
1998-1-9	0.9256
1998-1-10	1.0000

9.7.2　网络训练与测试

这里将每 3 天作为一个周期，3 天的股票数据作为网络的输入向量，输出则为预测日当天的股票价格。因此，输入层的神经元个数 N=3，输出层的神经元个数 M=1，样本个数 $K=L-(M+N)+1=7$ 个。为了提高网络精度，中间层的神经元个数设定为 50 个。网络创建代码为：

```
spread=1;
net=newrbe(P,T,spread);
```

其中，spread 为径向基函数的分布密度，spread 越大，网络的预测性能越平滑。但是

并不是越大越好，过大的 spread 可能导致计算上出现问题，这里先将其设定为 1。P 和 T 分别表示训练样本中的输入向量和目标向量。newrbe 可以创建一个准确的 RBF 网络，也就是说，网络的创建过程也就是训练过程，所创建网络的误差已经为 0 了。

这里将前 5 个样本作为训练样本，后 2 个样本作为测试样本。测试代码为：

```
Y=sim(net,P_test);
```

P_test 为测试样本向量。不断增加 spread 的值，观察它对最终输出的影响，结果如表 9-12 所示。可见，spread 越大，网络的输出就越接近于真实值。但当 spread 增加到 5 后，对输出的影响就没有那么明显了。因此，这里取 spread=5，这时的预测误差为 0.0948，考虑到训练样本的容量，这样的误差还是可以接受的。

表 9-12 测试结果

spread	输 出
1	0.7151
2	0.8054
3	0.8226
4	0.8283
5	0.8308



这里的训练样本容量太小，导致了网络的预报误差过大。如果想获得更好的预测性能，建议采用更大容量的训练样本，如 1 年甚至几年的股票数据。

本例完整的 MATLAB 代码为：

```
P=[0.1254 0.2315 0.3297;
   0.2315 0.3297 0.4376;
   0.3297 0.4376 0.5425;
   0.4376 0.5425 0.6078;
   0.5425 0.6078 0.7119]';
T=[0.4376 0.5425 0.6078 0.7119 0.8685];
P_test=[0.6078 0.7119 0.8685]';
net=newrbe(P,T,5);
Y=sim(net,P_test)
```

9.8 财务失败预测的 MATLAB 实现

财务失败又称财务困境，最严重的财务困境就是企业破产。当一个企业无能力履行合同、按时支付债权人利息和偿还本金时，该企业就面临财务失败。事实上，企业陷入财务困境直至破产是一个逐步的过程，大多数企业的财务失败都是由财务状况异常到逐步恶化，最终导致财务失败或破产的。因此，企业的财务失败不但具有先兆，而且是可预测的。随着中国加入 WTO，金融银行业的竞争日趋激烈，正确地预测企业财务失败对于保护投资者和债权人的权益、对于经营者防范财务危机、对于政府管理部门监控上市公司质量和证券

市场风险，都具有重要的现实意义。当前被广泛研究并应用于财务失败预测的模型主要有统计模型和人工智能模型两大类。

传统的统计模型包括多元判别分析模型MDA和对数回归模型 Logistics Regression 等，这两者也是应用最为广泛的模型。统计模型最大的优点在于其具有明显的解释性，存在的缺陷在于其过于严格的前提条件。如 MDA 要求数据分布服从多元正态分布、同协方差等；对数回归模型虽然对数据分布的要求有所降低，但仍对财务指标之间的多重共线性干扰敏感，而现实中大量数据分布都不符合这些假设前提，从而限制了统计模型在这一领域中的应用。

9.8.1 问题描述

随着信息技术的发展，人工智能和机器学习的一些分类和预测的算法也被引入到金融信用风险评估领域中来，主要包括人工神经网络和决策树的方法。决策树是一种自顶向下的分类方法，它通过对一组训练样本的学习，构造出决策型的知识表现。决策树具有速度快、精度高、生成模式简单等优点，但是这种归纳学习的方法容易造成模型的过度拟合，而且当问题复杂时，决策树的解释性也会降低。人工神经网络具有良好的容错性、自适应性和很强的泛化功能。现实世界中的企业财务失败预测问题往往非常复杂，企业的各项财务指标之间相互影响，呈现出复杂的非线性关系，而神经网络正是处理这类非线性问题的强有力的工具，近年来开始被引入到金融信用风险评估领域中。尤其是基于神经网络的企业破产预测方法逐渐显示出它的优越性，已经开始成为新的研究热点，应用的模型也从主要以 BP 网络为主逐渐扩展到其他类型的网络，本例尝试利用以竞争神经网络为基础的学习矢量量化 LVQ 网络，基于中国上市公司真实数据对财务失败进行预测。

9.8.2 样本的收集和处理

本例采用的用于构建破产预测模型的财务数据全部来自于国内上市公司的真实数据，其中选取的破产公司是指在连续两年内被股市特别处理（ST）的公司，同时依据行业分类选取该行业的其他公司为正常公司，以财务状况异常最早发生日为基准日，选取这些公司在基准日前两年的财务报表数据。共选取了 102 家财务状况异常公司，481 家正常公司（不同年份的同一家公司也认为是不同的公司），共 583 家公司来构建样本集合。出于篇幅限制的原因，这里选用了其中的 8 个非 ST 公司和 6 个 ST 公司的样本。

这里把所有样本数据分成两份，分别是训练集和测试集。已有研究表明，在分类模型的建立过程中，如果训练集合中两类样本数据的数量相当，则所建模型具有较强的健壮性，因此这里的训练集由相同数量的两类样本构成（ST 和非 ST 公司分别是 5 个样本）。测试集中 ST 和非 ST 公司样本分别是 1 和 3 个，用于测试在训练集上构建的 LVQ 网络模型的预测精度。

特征选择是模式识别分类问题中的关键步骤，综合考虑覆盖面和计算量等问题，这里选择了最能反映公司财务状况的财务比率指标的 5 个财务指标：净资产收益率、流动比率、股本权益比率、总资产增长率和营运资本率。样本数据如表 9-13 所示。

表 9-13 样本数据

公司类型	净资产收益率	流动比率	股本权益比率	总资产增长率	营运资本率
ST	0.25	0.31	0.27	0.19	0.12
ST	0.23	0.28	0.19	0.21	0.11
正常	0.75	0.61	0.75	0.50	0.50
ST	0.21	0.27	0.24	0.17	0.14
正常	0.78	0.60	0.68	0.50	0.52
正常	0.75	0.59	0.75	0.50	0.49
正常	0.81	0.60	0.71	0.50	0.50
ST	0.19	0.30	0.21	0.18	0.13
正常	0.81	0.60	0.72	0.50	0.57
ST	0.25	0.28	0.19	0.23	0.09
正常	0.73	0.55	0.67	0.50	0.61
ST	0.15	0.23	0.18	0.21	0.16
正常	0.72	0.61	0.73	0.50	0.62
正常	0.70	0.59	0.72	0.48	0.59

由于所有的数据都已经位于区间[0,1]中了，所以对于表 9-13 中的数据，无需进行归一化处理就可以直接投入训练及测试。

这里利用前 5 个 ST 公司的样本和前 5 个正常公司的样本作为网络的训练样本，后 1 个 ST 公司和后 3 个正常公司的样本作为网络的测试样本。

首先，创建一个 LVQ 网络：

```
net=newlvq(minmax(p),8,[0.5 0.5]);
```

其中，p 为训练样本中的输入向量，8 表示网络的竞争层神经元的数目，[0.5 0.5]表示输入样本中属于第一类的数据占 50%，属于另一类的占 50%，学习算法为 learnlv1。由于竞争层神经元的数目可以影响网络分类性能，因此需要通过不断实验进行选择。

接下来利用训练样本对网络进行训练：

```
tc=[1 1 2 1 2 2 2 1 2 1];
t=ind2vec(tc);
net=train(net,p,t);
```

其中，tc 为输入向量所属的类别，ST 公司用 1 表示，正常公司用 2 表示。函数 ind2vec 将类别向量转换为网络可用的目标向量。网络的训练结果为：

```
TRAINR, Epoch 0/100
TRAINR, Epoch 2/100
TRAINR, Performance goal met.
```

网络经过两次训练后，误差就达到了要求。训练结果如图 9-16 所示。

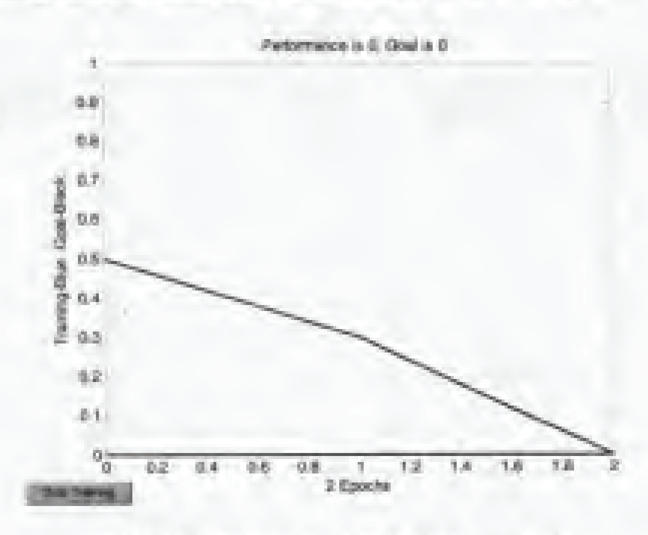


图 9-16 训练结果（竞争层单元：8 个）

对网络进行仿真，检验网络是否对训练数据中的输入向量进行了正确的分类。

```
y=sim(net,p);
yc=vec2ind(y)
```

输出结果为：

```
yc =
     1     1     2     1     2     2     2     1     2     1
```

yc =tc，可见网络的分类是正确的。

接下来对网络进行交叉检验，即利用网络对训练样本以外的数据进行分类：

```
tc_test=[2 1 2 2];
y_test=sim(net,p_test);
yc_test=vec2ind(y_test)
```

y_test 为网络测试样本中的输入向量，tc_test 表示输入向量的类别。输出为：

```
yc_test =
     2     1     2     2
```

yc_test=tc_test，可见网络对输入向量进行了成功分类。

9.9 农作物虫情预测的 MATLAB 实现

农作物的主要害虫常年对作物造成严重危害，使农业经济遭受严重损失。根据害虫的发生、发展规律，以及作物的物候和气象预报等资料，进行全面分析，做出其未来的发生期、发生量和危害程度等估计，预测害虫的未来发展动态，这项工作就叫做农作物虫情预测。虫情预测工作是进行害虫综合防治的必要前提，只有对害虫发生危害的预测做得及时准确，才可以正确地拟定综合防治计划，及时采取必要的措施，经济有效地压低害虫的发生数量，保证农作物的高产和稳产。

按照预测内容来划分，虫情预测可以分为发生期预测、发生量预测、迁飞害虫预测和灾害程度预测，以及损失估计等 4 种；按照预测时间长短划分，可以分为短期预测、中期预测和长期预测等 3 种。



我国关于虫情预测问题的研究，起步较早，目前主要采用以下3种方法：

（1）统计法。根据多年的积累资料，探讨某种因素，如气候因素和物候现象等，与害虫某一虫态的发生期和发生量之间的关系，或害虫种群本身前后不同的发生期和发生量之间的相关关系，进行相关回归分析或数理统计计算，构建各种不同的预测模型。

（2）实验法。应用生物学方法，主要求出各虫态的发育速率和有效积温，然后应用气象资料预测其发生期。另一方面，利用实验方法探讨营养、气候和天敌等因素对害虫生存和繁殖能力的影响，为害虫发生量的预测提供依据。

（3）观察法。直接观察害虫的发生和作物物候的变化，明确虫口密度、生活史与作物生育期的关系。应用物候现象、发育进程、虫口密度和虫态历期等观察资料进行预测，是我国目前最通用的预测方法。该方法主要可以预测发生期、发生量和灾害程度。

9.9.1 基于神经网络的虫情预测原理

众所周知，虫害的发生和自然因素之间有着密切的联系，它同时受气温、日照和降雨量等因素的影响。影响虫害发生量的各因子之间存在复杂的相互作用。由于自身的缺点，利用传统方法很难建立起一个精确和完善的预测模型。而 BP 神经网络具有对非线性复杂系统预测的良好特性，可以有效地描述其本身具有的不确定、多输入等复杂的非线性特性。

从影响虫害发生量的气候因子角度来说，虫害发生量主要受以下3个因素制约：

（1）温度。昆虫是变温动物，体温基本上是随着外界温度的变化而变化的。而外界温度的变化直接影响着昆虫代谢率的高低，从而直接影响昆虫的生长发育、繁殖和生存等生命活动行为。昆虫对外界温度的变化适应不是无限的，而是有一定的适应范围。每种昆虫都有一定的温度适应范围，超过这一温度范围，昆虫的繁殖就会停止甚至死亡。了解每种害虫对温度的适应范围，对于分析和预测害虫的发生期和发生量有着重要意义。

（2）湿度和降雨。湿度和降雨可以直接影响昆虫的生长发育和生存。外界环境的湿度和降雨都是通过影响昆虫体内的含水量而产生作用的。所以，当外界环境的影响使得虫体内的水分调节失去平衡时，便可引起昆虫生长发育和繁殖等方面的反常表现。

（3）光。对于昆虫来说，光并不是一种生存条件，但外界光因素与昆虫的趋向性、活动行为和生活方式等都有着直接或间接的联系。

自然界中的各个气候因素是相互影响、综合作用于昆虫的。在进行虫情预测时，不能够只根据某一项单个指标，而是要注意综合作用影响。

9.9.2 BP 网络设计

本实例的预测对象是我国某地的田间水稻。水稻螟虫是水稻的重要害虫之一，尤其是二化螟。从温度上说，二化螟的发生发展和温度的变化关系十分密切，二化螟的抗低温能力较强，抗高温能力较弱，适宜温度在16～30摄氏度之间，35度以上的高温就容易使二化螟死亡。从降雨量角度讲，一方面，大量的降雨会导致气温下降，有利于二化螟的生存；另一方面，充沛的降雨会淹死大量的幼虫。因此，降雨对二化螟的影响是比较复杂的，需要综合考虑。

BP网络的输入和输出层的神经元数目，是由输入和输出向量的维数确定的。输入向量的维数也就是影响因素的个数，这里综合考虑了影响虫情的各种因素，选取了平均气温、最低气温、日照时间和降雨量等4个因素，所以输入层的神经元个数为4。为了细化虫害的等级，这里将虫害发生量分为4级，目标输出模式为（0 0 0 1）、（0 0 1 0）、（0 1 0 0）和（1 0 0 0），分别对应1级、2级、3级和4级。因此，输出层神经元的个数也为4。由于输出向量的元素为0-1值，因此，输出层神经元的传递函数为可选用S型对数函数logsig。

实践表明，隐含层数目的增加可以提高BP网络的非线性映射能力，但是隐含层数目超过一定值，网络性能反而会降低。而单隐层的BP网络可以逼近一个任意的连续非线性函数。因此，这里采用单隐层的BP网络。隐含层的神经元个数直接影响着网络的非线性预测性能。这里根据Kolmogorov定理，设定网络的隐含层神经元个数为9。按照一般的设计原则，隐含层神经元的传递函数为S型正切函数tansig。

网络结构确定后，需要利用样本数据通过一定的学习规则进行训练，提高网络的适应能力。学习速率是训练过程的重要因子，它决定每一次循环中的权值变化量。在一般情况下，倾向于选择较小的学习速率保证学习的稳定性，这里取学习速率为0.05。

本实例所使用的数据为该地区的田间水稻从1996年到2003年间的5月到10月的虫害发生程度及相应的气象数据。本来应该取1996年到2002年之间的数据作为网络的学习训练样本，2003年的数据作为预测样本，但这里限于篇幅原因，在利用神经网络工具箱进行编程时，只利用2000~2002年的数据作为训练样本，2003年的数据作为预测样本。这样做的直接后果会导致网络的预测精度下降，但这里我们更关心的是演示利用神经网络工具箱进行虫情预测的全过程。样本数据如表9-14所示。

表9-14　归一化后的样本数据

年　份	月	平均气温	最低气温	日照时间	降雨量	虫害程度
2000	5	-0.0909	-0.1408	-0.2500	-0.2984	0 0 0 1
	6	0.4825	0.3944	0.1250	0.3037	0 0 0 1
	7	0.9580	0.9718	0.9688	-0.7801	1 0 0 0
	8	0.6643	0.7183	0.5000	0.0419	1 0 0 0
	9	0.0350	0.0423	0.0000	-0.3665	0 0 0 1
	10	-0.6224	-0.6620	-0.0625	-0.8796	0 0 0 1
2001	5	-0.2727	-0.7324	0.5625	-0.7277	0 0 0 1
	6	-0.0909	0.0000	0.8125	-0.6073	0 1 0 0
	7	0.9580	1.0000	0.6875	0.0733	1 0 0 0
	8	0.8601	0.9296	0.2812	-0.3979	1 0 0 0
	9	0.0909	0.0141	0.1563	-0.4660	1 0 0 0
	10	-0.9860	-1.0000	-0.5625	-0.4241	0 0 0 1
2002	5	0.1189	-0.1127	0.6250	-0.6021	0 0 0 1
	6	0.3706	0.3521	0.0313	-0.6073	0 0 0 1
	7	0.6923	0.7324	-0.3125	0.2670	0 0 1 0
	8	0.6643	0.7324	-0.0625	0.1361	1 0 0 0
	9	-0.0350	0.0423	-0.5313	-0.8482	1 0 0 0
	10	-0.4266	-0.5070	-0.1250	-0.8586	0 1 0 0

（续表）

年份	月	平均气温	最低气温	日照时间	降雨量	虫害程度
2003	5	0.0490	0.0000	-0.0937	-0.0995	0 0 0 1
	6	0.2587	0.3662	-0.5313	1.0000	0 0 1 0
	7	0.7203	0.8028	-0.1875	0.4346	0 0 1 0
	8	0.9301	0.9014	0.9688	-0.8691	1 0 0 0
	9	0.3287	0.3239	0.2813	-0.6702	0 0 0 1
	10	-0.6084	-0.5211	-0.2813	-0.4346	0 0 0 1

该实例完整的 MATLAB 代码为：

```
%构建训练样本中的输入向量p
p1=[-0.0909 0.4825 0.9580 0.6643 0.0350 -0.6224;
    -0.1408 0.3944 0.9718 0.7183 0.0423 -0.6620;
    -0.2500 0.1250 0.9688 0.5000 0.0000 -0.0625;
    -0.2984 0.3037 -0.7801 0.0419 -0.3665 -0.8796];
p2=[-0.2727 -0.0909 0.9580 0.8601 0.0909 -0.9860;
    -0.7324 0.0000 1.0000 0.9296 0.0141 1.0000;
    0.5625 0.8125 0.6875 0.2812 0.1563 -0.5625;
    -0.7277 -0.6073 0.0733 -0.3979 -0.4660 -0.4241];
p3=[0.1189 0.3706 0.6923 0.6643 -0.0350 0.4266;
    -0.1127 0.3521 0.7324 0.7324 0.0423 -0.5070;
    0.6250 0.0313 -0.3125 -0.0625 -0.5313 -0.1250;
    -0.6021 -0.6073 0.2670 0.1361 -0.8482 -0.8586];
p=[p1 p2 p3];
%构建训练样本中的目标向量t
t1=[0 0 1 1 0 0;
   0 0 0 0 0 0;
   0 0 0 0 0 0;
   1 1 0 0 1 1];
t2=[0 0 1 1 1 0;
   0 1 0 0 0 0;
   0 0 0 0 0 0;
   1 0 0 0 0 1];
t3=[0 0 0 1 1 0;
   0 0 0 0 0 1;
   0 0 1 0 0 0;
   1 1 0 0 0 0];
t=[t1 t2 t3];
%创建一个BP网络，隐含层有9个神经元，传递函数为tansig
%中间层有4个神经元，传递函数为logsig，训练函数为trainlm
net=newff(minmax(p),[9,4],{'tansig','logsig'},'trainlm');
%训练步数为50
%目标误差为0.01
net.trainParam.epochs=50;
net.trainParam.goal=0.01;
net=train(net,p,t);
```

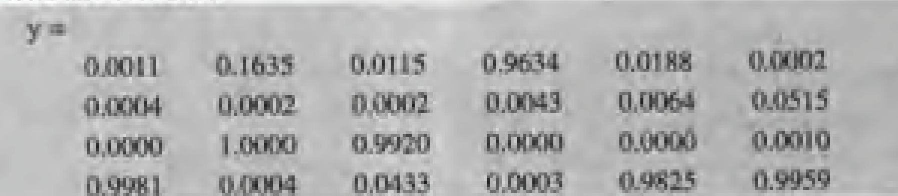
```
%预测 2003 年的虫情
p_test=[0.0490 0.2587 0.7203 0.9301 0.3287 -0.6084
        0.0000 0.3662 0.8028 0.9014 0.3239 -0.5211
        -0.0937 -0.5313 -0.1875 0.9688 0.2813 -0.2813
        -0.0995 1.0000 0.4346 -0.8691 -0.6702 -0.4346];
y=sim(net,p_test);
```

运行以上代码，可以得到网络的训练结果为：

```
TRAINLM, Epoch 0/50, MSE 0.467963/0.01, Gradient 3.20693/1e-010
TRAINLM, Epoch 13/50, MSE 0.00518824/0.01, Gradient 0.308827/1e-010
TRAINLM, Performance goal met.
```

可见网络经过 13 次训练后即可达到误差要求，结果如图 9-17 所示。

网络的输出结果为：



```
y =
    0.0011    0.1635    0.0115    0.9634    0.0188    0.0002
    0.0004    0.0002    0.0002    0.0043    0.0064    0.0515
    0.0000    1.0000    0.9920    0.0000    0.0000    0.0010
    0.9981    0.0004    0.0433    0.0003    0.9825    0.9959
```

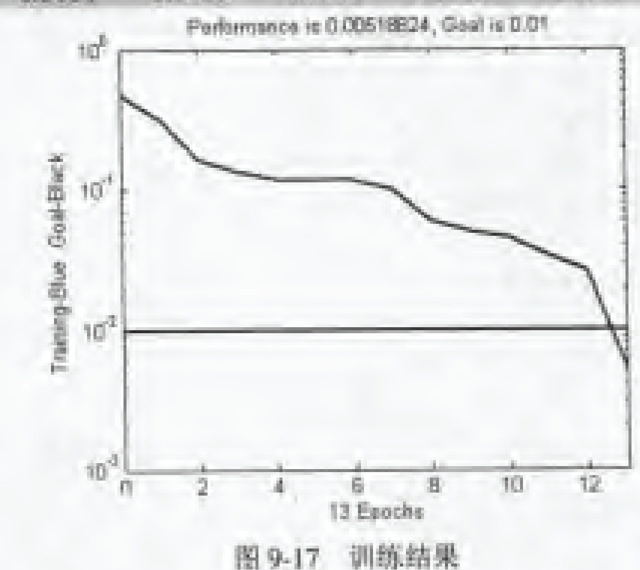


图 9-17　训练结果

通过与实际虫情相比较，可得到网络的预测误差为 0.1690。考虑到样本容量比较小的原因，这已经是可以接受的结果了。网络的预报误差曲线如图 9-18 所示。

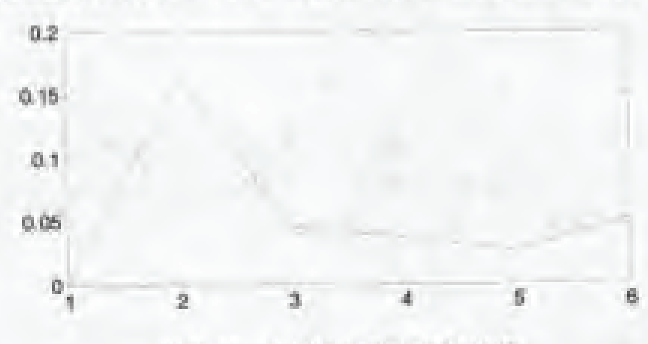


图 9-18　网络的预测误差曲线

9.10 小　结

本章以电力负荷预报、河道浅滩演变预测、地震预报、交通运输能力预测、股市预测、财务失败预测和虫情预测等 7 个实例为对象，研究了神经网络及神经网络工具箱在工业系统、地球物理、经济领域和农业工程中的应用。这些问题有一个共同的特点，即都是受多个影响因素共同影响，且各个因素之间有着比较复杂的关系，具有高度不确定的非线性系统，利用传统的预测方法有着诸多限制，而采用神经网络方法则能较好地克服这些限制，实现精确的非线性预测。

为了进一步提高网络的预测精度，需要从以下几个方面展开研究：

（1）网络的训练过程是从给定的样本数据中归纳出输入、输出之间的复杂规律，为了能够更加准确地对系统进行预测，样本数据应该尽可能准确。

（2）提高网络预测能力的主要途径有：1）尽可能增加样本的涵盖面；2）在输入中尽可能地包括影响输出的主要因子；3）确定适当的收敛误差。对于常用的 BP 算法，可考虑采用遗传算法、小波分析和径向基函数做进一步的深入研究。

（3）针对 BP 算法中存在的收敛速度慢、易陷入局部最小值的问题，可采用附加动量法和自适应学习速率法在一定程度上解决这些问题。附加动量法是在 BP 算法的基础上，在每个权值变化上加上一项正比于上一次权值变化量的值，并根据 BP 算法来产生新的权值变化，利用附加动量法可能会避开某些局部最小值。自适应学习速率法是在学习过程中不断修正学习速率，有利于提高学习效率，缩短学习时间。



第 10 章　基于神经网络的模糊控制

模糊控制器自 1974 年问世以来，其基本设计方法一直没有大的变化。它的基本设计思想就是借助人工操作的经验，通过合成关系，把输入的模糊量和关系矩阵合成，推导出控制量。近年来，随着神经网络理论的迅速发展，一些研究人员尝试将神经网络引入到模糊控制器中，取得了不错的效果，控制器性能得到了进一步的提高。可以说，将模糊控制和神经网络两者结合组成的神经网络模糊控制技术是控制理论研究者们关注的焦点之一。

本章的主要内容包括：

- 基于神经网络的模糊控制原理
- 神经模糊控制器的 MATLAB 实现

10.1　引　　言

传统的模糊控制器是利用模糊逻辑来模仿人的思维，对那些非线性时变的复杂系统，以及无法建立数学模型的系统实现控制的。它的设计过程为定义由语言变量描述的模糊子集，获取控制规则，并对这些规则进行模糊化和去模糊化。模糊规则的品质决定了控制系统性能的优劣。在传统的模糊控制系统中，控制规则的获取完全按照设计者和操作人员的经验进行。在某些系统中，由于设计者和操作人员经验的局限性，造成模糊控制器的设计无法保证最优或次优的控制性能。而神经网络具有自学习的能力，通过训练来学习给定的经验，并由此生成模糊控制规则。

将神经网络与模糊控制相结合，为模糊控制器提供了良好的学习功能，并自动生成模糊控制规则。通过神经网络实现的模糊控制，对于知识的表达并不是通过显式的一条条的规则，而是把这些规则隐含地分布在整个网络之中。在控制应用时，不必进行复杂的规则搜索和推理，只需通过高速并行的分布计算就可产生输出结果。

10.2　神经网络模糊控制的结构和特征

神经网络模糊控制器是由神经网络和模糊控制共同组成的混合系统，这种混合系统兼备这两种技术的优点。本节将详细介绍该系统的结构及特征。

10.2.1　神经网络模糊控制器的结构

神经网络模糊控制器的结构一般有 3 种组合形式，如图 10-1 所示。这 3 种组合形式分别对应神经网络在模糊控制的 3 种作用。

- 利用神经网络驱动模糊推理；
- 利用神经网络记忆模糊控制规则；
- 利用神经网络优化模糊控制器的参数。

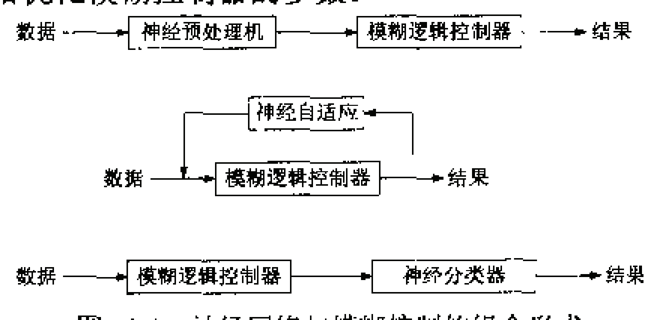


图 10-1 神经网络与模糊控制的组合形式

如图 10-2 所示，为一种带有神经网络的模糊控制器结构框图。这种结构是利用联想记忆神经网络来记忆模糊控制规则的。

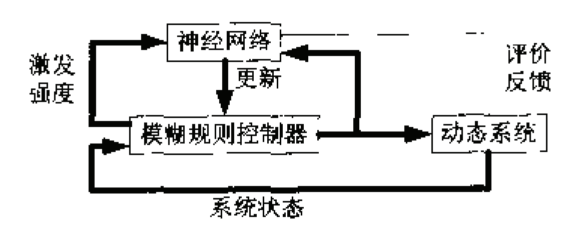


图 10-2 神经模糊控制器结构框图

这种模糊控制器的控制规则为：如果 E=A_i，dE=A_i，那么 U=C_i。其中，E 和 dE 为两个输入变量，如误差和误差变化率，分别对应神经网络的两个输入结点，结点个数由输入语言变量的级数决定。U 为输出变量，如校正值，对应着神经网络的所有输出结点，输出结点的个数是由输出语言变量的级数决定的。如果通过 BP 学习算法训练神经网络，调整网络的权值，使网络能够实现样本的输入/输出的对应映射，那么这种多层神经网络还应该有隐含层，隐含层神经元的个数需要通过反复实验确定。

10.2.2 神经网络模糊控制器的特征

以 BP 网络模糊控制器为例，有一个单隐层的多层前馈 BP 网络，结构如图 10-3 所示。设 $X_k=[x_{0k},x_{1k},\cdots,x_{nk}](x_{0k}=1)$ 为网络第 k 个输入的样本，则第 j 个隐含层神经元的输出 z_{jk} 可表示为：

$$z_{jk}=\varphi\left(\sum_{i=1}^{n}w_{ij}x_{ik}\right)$$

其中，w_{ij} 为第 i 个输入神经元和第 j 个隐含层神经元之间的连接权值，φ 为 S 型函数，定义为 $\varphi(x)=1/(1+e^{-x})$。

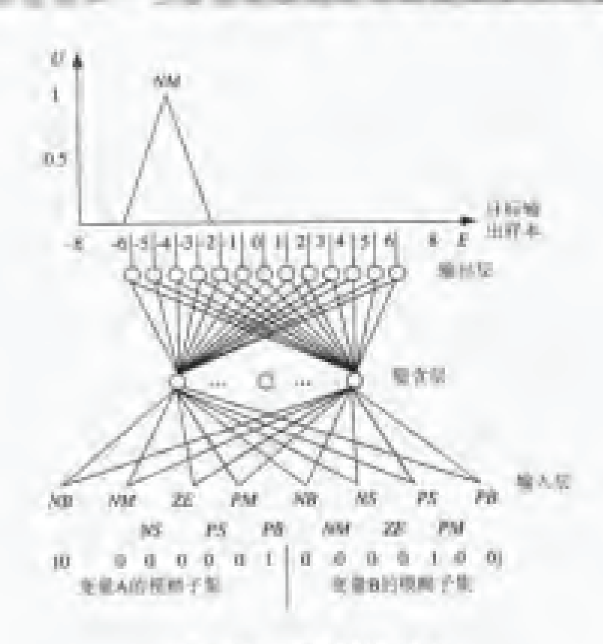


图 10-3　BP 网络结构

为了训练网络，建立模糊关系，需要借助数值样本来描述输入/输出模糊子集。假定有两个输入变量 A 和 B，一个输出变量 C，都是由如图 10-4 所示的模糊子集来定义的，模糊规则如表 10-1 所示。网络输入空间对应于变量 A、B，被划分为两个部分。由于网络的每个单元都对应着输入变量的某个模糊子集，因此，每个输入变量都有 7 个输入单元与其 7 个模糊子集相对应。

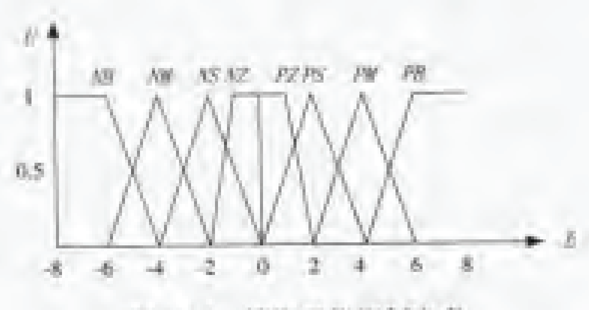


图 10-4　模糊子集规划定义

表 10-1　模糊规则表

<table>
<tr><th>A \ B</th><th>PB</th><th>PM</th><th>PS</th><th>ZE</th><th>NS</th><th>NM</th><th>NB</th></tr>
<tr><td>PB</td><td colspan="2">NS</td><td colspan="2">NM</td><td colspan="3">NB</td></tr>
<tr><td>PM</td><td colspan="2">PS</td><td rowspan="2">PS</td><td rowspan="2">NS</td><td rowspan="3">NM</td><td colspan="2" rowspan="2">NB</td></tr>
<tr><td>PS</td><td colspan="2" rowspan="2">PM</td></tr>
<tr><td>ZE</td><td rowspan="2"></td><td>ZE</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>NS</td><td colspan="2"></td><td>PS</td><td>NS</td></tr>
<tr><td>NM</td><td colspan="3" rowspan="2">PB</td><td rowspan="2">PM</td><td>PS</td><td colspan="2">NS</td></tr>
<tr><td>NB</td><td>PM</td><td colspan="2">PS</td></tr>
</table>

网络的输入信号格式为：

$$\left[\mu_{NB}(a),\mu_{NM}(a),\mu_{NS}(a),\mu_{ZE}(a),\cdots,\mu_{ZE}(b),\mu_{PS}(b),\mu_{PM}(b),\mu_{PB}(b)\right]$$

网络的每一个输出单元都对应着输出变量空间中的一个量化值。因此，输出的模糊子集就可用量化空间上的隶属函数来表示。输出信号的格式为：

$$\left[\mu_y(c_1),\mu_y(c_2),\cdots,\mu_y(c_{m-1}),\mu_y(c_m)\right]$$

变量 c 的论域分为 13 档，即{-6,-5,…,5,6}。根据上述定义，可将控制规则“若 A 是 PB 且 B 是 PS，则 C 为 NM”的输入信号表示为：

$$[0,0,0,0,0,0,1;0,0,0,0,1,0,0]$$

其中，$\mu_{NB}(a)=1$，$\mu_{PS}(b)=1$，其余均为 0。输出信号为：

$$[0,0.5,1,0.5,0,0,0,0,0,0,1,0,0]$$

其中，$\mu_{NM}(-5)=0.5$，$\mu_{NM}(-4)=1$，$\mu_{NM}(-3)=0.5$，其余均为 0。

出此可见，所有的控制规则都可用一系列的输入/输出数字信号来表示。BP 算法用于网络训练，使输入信号对应于目标输出值。因此，经过训练的网络就相当于一个模糊关系存储器。

如果想通过训练好的网络推导出其他结论，只要把输入变量的实际值模糊化后再输入网络，如 a=2.5，b=－0.5，则模糊化后的输入信号为：

$$[0,0,0,0,0.75,0.25,0,0.5,0.5,0,0,0,0]$$

在网络的输出层就会得到一个输出模糊子集，再经过解模糊子集可得到真实的输出值。

经过训练的网络能够对输入训练数据准确地做出响应。由于网络信息的存储结构十分巧妙，因此，它激发若干条规则的响应情况是很难准确描述的。一般来说，网络的推理机构受以下因素的影响：

（1）一般规律。如果输入模糊子集接近于网络所用的训练模糊子集，则输出几乎和该条训练规则的结果相同。

（2）非线性插值规律。当输入信号和训练用的模糊子集有较大差异时，输入后就有一系列的相关结点被激发，输出量就是这些被激发规则的非线性插值。一条规则的输出响应和当前输入与训练该规则时所用输入的距离成反比。

10.2.3 神经网络模糊控制器的应用实例

下面通过介绍一个采用神经网络的自组织模糊控制器对神经网络模糊控制器进行演示。自组织模糊控制器（Fuzzy Logic Self-organizing controller，FLSOC）的系统结构如图 10-5 所示。

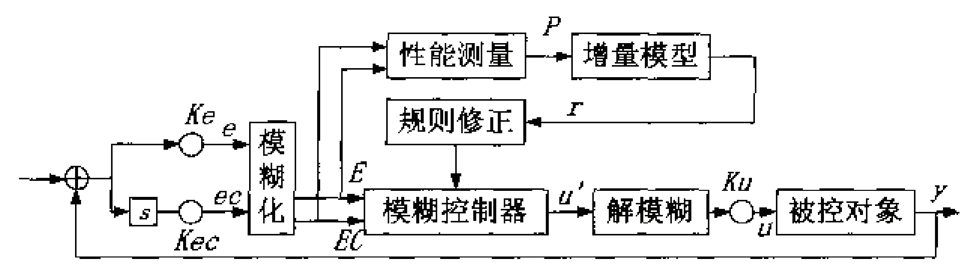


图 10-5　FLSOC 系统结构

FLSOC 通常包括以下几个部分：

- 一个基于模糊规则推理算法模糊控制器；
- 用于对控制自身作用进行评估的性能测量部分；
- 一个增量处理模型，用于修改模型控制器中的规则，以对控制作用给出增补。

基于神经网络的自组织模糊控制器（NNSOC）的系统结构如图 10-6 所示。其控制规则通过一个神经网络执行，而规则修正则由 BP 学习算法实现。组成 NNSOC 中的性能测量和处理模型部分与 FLSOC 中的类似，下面仅介绍其不同之处。

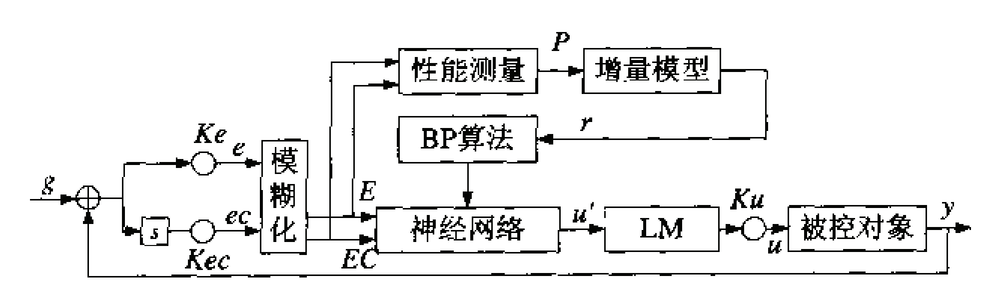


图 10-6　NNSOC 系统结构

1. 控制器

FLSOC 中采用的模糊控制器是由一组模糊规则和基于模糊推理合成算法的推理机组成的。而 NNSOC 中的控制器是由神经网络实现的，其输入编码格式和上小节所述相同，网络输出代表了一个真实的控制量，而不是一个模糊子集，因此无需了解模糊的过程。在 FLSOC 中，采用比例因子 K_e、K_{ce} 和 K_u 的目的是使输入/输出量的变化范围折算到相应模糊子集的定义领域上。因此，网络的输出 u' 始终位于区间[0,1]中，并与控制对象的输入之间为线性关系。

2. 控制器自校正

在 FLSOC 中，规则修正过程为：

（1）确定造成当前不良结果的控制行为。

（2）确定该控制行为产生于哪一条规则。

（3）由输出偏移计算输入增补量。

（4）修正相应规则，以使得在类似情况时增强控制作用。

对于 FLSOC 来说，规则修改可以是修改关系矩阵，也可以是修改相应规则。对于前者来说，由于关系矩阵通常容量很大，对它进行搜索与修改需要很多时间。对于后者来说，通常是由生成的新规则替代原来规则。但往往由于每条旧规则的修改会由多于 1 条，甚至

是 3 条新规则所替代，这样会导致规则总数急剧增多。随着控制器规则数目不断增加，用于控制行为评估的计算时间也将相应增加。

在 NNSOC 中，可以通过调整网络的权值来修正控制量。采用 BP 算法进行一定次数的反复计算，就可以修正网络的权值而收到期望效果。NNSOC 的学习过程具有以下优点：

（1）它既不需要对控制的输出解模糊，也不需要对输入变量模糊化。

（2）由于网络的前向推算时间是固定的，因此，估算控制行为的时间也将不变。

（3）如果可以研制出具有在线学习能力的神经网络芯片，则就有可能实现控制工程的高速处理，很大程度上提高实时性。

在 NNSOC 形成一套成功的控制策略之前，需要进行若干次训练，每次训练都由一批预定数量的采样值完成，权值的修改是在训练过程中的每个采样时刻进行的。修正后的权值被用于下一次训练，并且每次训练前均应该对控制变量初始化。每次训练使系统的响应特性（上升时间、超调量等）逐渐得到改善。不断地重复训练，直到 NNSOC 的性能可以接受为止。

在给定的采样时刻，采用 BP 算法，其反复计算的次数取决于网络的大小。通常，大型网络需要更多的学习次数。但是，如果在每一采样时刻，计算的次数太多，则网络有可能会忘记以前学习过的内容。可见，在 NNSOC 中，计算次数的选择也极为重要。

例 10.1 假定有一个二阶系统，状态方程为：

$$\begin{cases} \dot{X} = AX + Bu(t) \\ y(t) = CX \end{cases}$$

其中，$X = [x_1, x_2]^T$ 为状态变量，$u(t)$ 为外界输入变量，$y(t)$ 为输出变量，系数矩阵 $A = \begin{bmatrix} 0 & 1 \\ a_2 & a_1 \end{bmatrix}$，$B = \begin{bmatrix} 0 \\ a_3 \end{bmatrix}$，$C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$。令 $a_1 = -1.5$，$a_2 = -0.5$，$a_3 = 10$，目标输出为 $s(t) = 10$。控制器有两个输入变量，分别为误差 $e(t) = s(t) - y(t)$ 和误差变化率 $ec(t) = de(t)/dt$，以及一个输出变量 $u(t)$。变量 EC 的模糊子集定义如图 10-7 所示。为了使进行目标值输出时的控制更为精确，变量 E 增加了两个模糊子集 PZ 和 NZ。

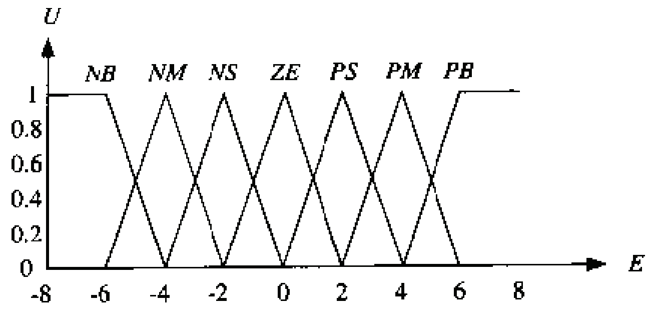


图 10-7 EC 的模糊子集定义

这里的控制器采用一个单隐层的BP网络，它的输入层有15个神经元，其中前8个代表$e(t)$的模糊子集，后7个代表$ec(t)$的模糊子集。隐含层有8个神经元，输出层只有1个神经元。这里取K_e=3，K_{ec}=30，网络输出u'在(0,1)之间取值，并且与实际控制对象的输入u成线性关系，由于此处u位于区间[-1,1]中，所以$u=2u'-1$。

假定“$k-1$”时刻的控制作用决定了控制器在k时刻的输出量，在采样时刻k修正环节的输入变化量为$r(k)$，并且“$k-1$”时刻的输入信号v_{k-1}的目标设定为$u(k-1)+Kr(k)$，这里取K=0.16是把信号$r(k)$从区间[-6,6]折算到[-1,1]上的一个比例常数。为了得到期望值，在每一个采样时刻利用BP算法计算5次来修正网络权值。

控制器的学习过程要通过若干次训练，每次训练采用400个样本数据，训练前要对控制变量初始化，学习训练一定要持续到获得满意的输出响应为止。通过选择不同的训练次数和选择不同的a_i的值，进行多次实验，结果表明：

（1）网络的训练次数越多，控制器的输出越稳定。

（2）同样结构的控制器，可适用于许多不同的被控对象，仅仅响应特性有所不同而已。

（3）当比例因子的变化范围很大时，也不会影响到该控制器的自组织性能。因此，该控制器具有很强的适用性。

10.3　基于MATLAB的神经模糊控制洗衣机仿真

20世纪90年代初期，日本松下公司推出了神经模糊控制全自动洗衣机。这种洗衣机能够自动判断衣物的质地软硬程度、衣量多少、脏污程度和性质等，应用神经模糊控制技术，自动生成模糊控制规则和隶属度函数，预设洗衣水位、水流强度和洗涤时间，在整个洗衣过程中实时调整这些参数，达到最佳的洗衣效果。

10.3.1　洗衣机的模糊控制

洗衣机的主要被控参量为洗涤时间和水流强度，而影响这一输出参量的主要因子是被洗物的浑浊程度和浑浊性质，后者可用浑浊度的变化率来描述。在洗涤过程中，油污的浑浊度变化率小，泥污的浑浊度变化率大。因此，浑浊度及其变化率可以作为控制系统的输入参量，而洗涤时间和水流强度可作为控制量，即系统的输出。实际上，洗衣过程中的这类输入和输出之间很难用一定的数学模型进行描述。系统运行过程中具有较大的不确定性，控制过程在很大程度上依赖操作者的经验，这样一来，利用常规的方法进行控制难以奏效。但是，如果利用专家知识进行控制决策，往往容易实现优化控制，这就是在洗衣机中引入模糊控制技术的主要原因之一。

根据上述的模糊控制基本原理，可得出确定洗涤时间的模糊推理框图如图10-8所示。其输入量为洗涤水的浑浊度及其变化率，输出量为洗涤时间。考虑到适当的控制性能需要和简化程序，定义输入量浑浊度的模糊词集为{清、较浊、浊、很浊}；定义浑浊度变换率的模糊词集为{零、小、中、大}；定义输出变量洗涤时间的模糊词集为{短、较短、标准、长}。描述输入/输出变量的词集都具有模糊特性，可以用模糊集合表示。因此，模糊概念

的确定问题就直接转化为求取模糊集合的隶属函数问题。

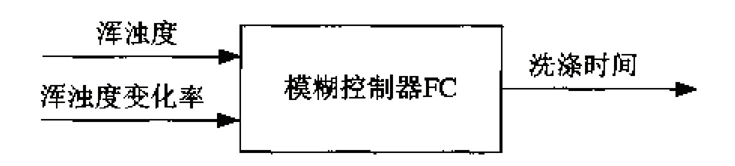


图 10-8　确定洗涤时间的模糊推理框图

通常定义一个模糊子集，实际上就是确定模糊子集隶属函数形状的过程。将确定的隶属函数曲线离散化，就得到了有限个点上的隶属度，构成了一个相应的模糊子集。如图 10-9 所示，定义的隶属函数曲线表示论域 x 中元素 x 对模糊变量 A 的隶属程度。设定该隶属函数的论域 x 为：

$$x=(-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6)$$

则有：

$$\mu_A(2)=\mu_A(6)=0.2,\quad \mu_A(3)=\mu_A(5)=0.7,\quad \mu_A(4)=1$$

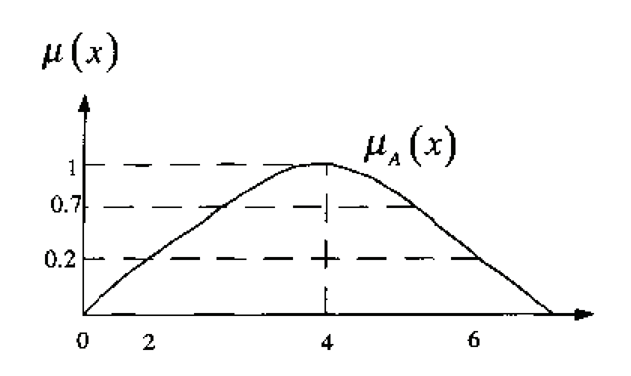


图 10-9　模糊变量 A 的隶属度函数

论域 x 中除了 2,3,4,5,6 外，各点的隶属度均为 0。那么模糊变量 A 的模糊子集为 A=0.2/2+0.7/3+1/4+0.7/5+0.2/6。

通过这个例子可以看出，在隶属函数的曲线确定后，就可以很容易地定义出一个模糊变量的模糊子集。洗衣机模糊控制的输入和输出变量的隶属函数如图 10-10 所示，由此可以相继确定它们的模糊子集。

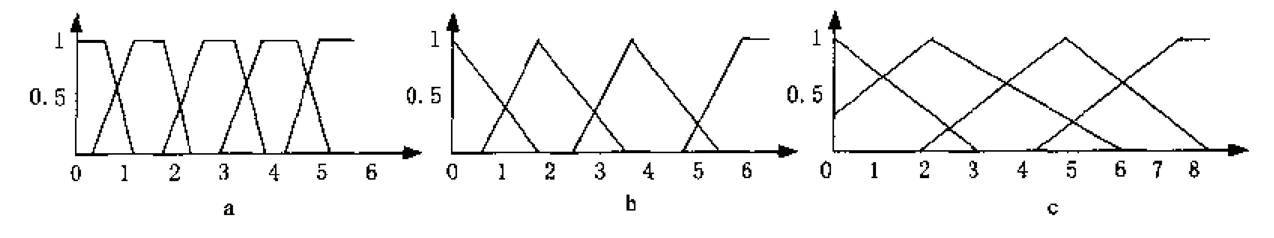


图 10-10　洗衣机模糊变量隶属函数（a.浑浊度；b.浑浊度变化率；c.洗涤时间 t）

洗衣机的模糊控制规则可以归纳 16 条，如表 10-2 所示。

表 10-2 洗衣机的模糊控制规则表

洗涤时间		浑浊度			
		清	较浊	浊	很浊
变化率	零	短	较短	标准	标准
	很小	标准	标准	标准	标准
	中	标准	长	长	长
	大	标准	标准	长	长

10.3.2 洗衣机的神经网络模糊控制器的设计

洗衣机模糊控制的控制部分框图如图 10-11 所示，模糊控制器如图中虚线所示。模糊控制的过程是这样的，首先洗衣机获取的浑浊度信息由传感器送到信息处理单元，分为浑浊度和浑浊度变化率送入模糊控制器。对于输入的模糊量，需要将其转换成模糊变量，通过单片机，利用查表法按照模糊推理法则做出决策，结果被认为是模糊变量，经过去模糊化单元处理，再由执行机构去修改洗涤时间，这样就完成了一次模糊控制算法过程。

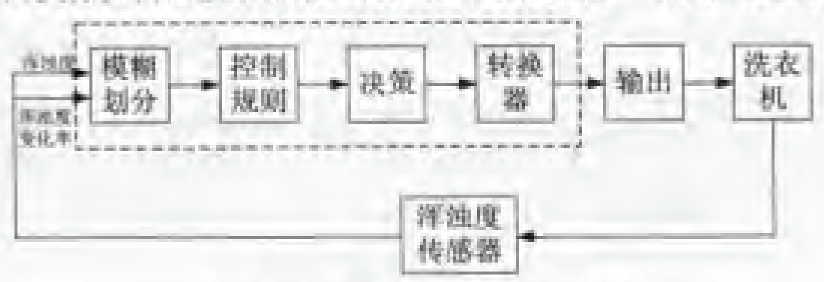


图 10-11 模糊控制的控制部分框图

一般的模糊控制洗衣机将“专家经验”通过模糊控制规则表体现出来，运行中通过查表做出控制决策，这比需要操作者设定程序的电脑控制洗衣机前进了一大步。但是，这种洗衣机由于规则表需要占用大量的内存空间，查表反应速度慢，只能够按照已经编入的规则进行控制，因此不够理想。而把神经网络和模糊控制相结合，则能够解决这些问题。

洗衣机的神经网络模糊控制是利用离线训练好的网络，通过在线计算即可得到最佳输出。这种控制模式的反应速度快，而且神经网络又具有自学习功能和联想能力，对于未在训练中出现的样本，也可以通过联系记忆的功能，做出控制决策，表现非常灵活。

洗衣机的神经网络模糊控制器的控制系统中含有多个神经模糊环节，下面仅介绍以浑浊度和浑浊度变化率为输入参量来确定洗涤时间的控制器。控制器的控制框图如图 10-12 所示，浑浊度神经网络结构如图 10-13 所示。

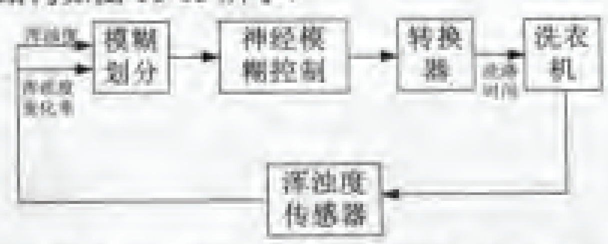


图 10-12 神经网络模糊控制器的控制框图

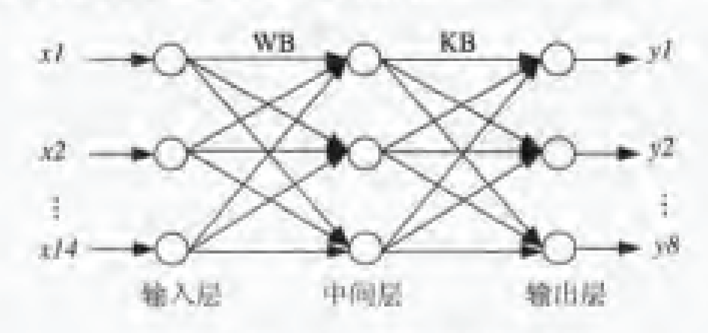


图 10-13　浑浊度神经网络结构

神经模糊控制器在输入/输出参量的选择，以及模糊论域和模糊子集的确定方面，与一般模糊控制器没有什么区别，只是在推理手段上引入了神经网络。令 x_1~x_7 为输入量浑浊度的模糊子集，x_8~x_{14} 为输入量浑浊度变化率的模糊子集，y_1~y_8 为输出控制量的模糊子集。从模糊控制规则表 10-2 可以看出，共有 16 条控制规则，每条规则都是一对样本，则共有 16 对样本。例如，当浑浊度为“清”，浑浊度变化率为“零”时，洗涤时间应该为“短”，这个样本可以表示为：

$$x=[1,0.6,0.1,0,0,0,0,1,0,0.5,0,0,0,0,0]^T\text{，}\quad y=[1,0,0.5,0,0,0,0,0,0]^T$$

其中，x 中的各元素为对应的隶属函数，即模糊子集的赋值。同理可列出其他 15 个样本对，并将它们依次送入神经网络进行离线训练，当训练结束后，神经网络已经记忆了模糊控制规则，使用时具有联想记忆功能。如表 10-3 所示，为每一个输入参量的模糊量。

表 10-3　输入参量的模糊量

输入参量		模糊量
浑浊度	清	1 0.6 0.1 0 0 0 0
	较浊	0 0.6 0.6 0 0 0 0
	浊	0 0 0.6 1 0 0 0
	很浊	0 0 0 0 1 0.6 0
浑浊度变化率	零	1 0.5 0 0 0 0 0
	很小	0 0.5 1 0.4 0 0 0
	中	0 0 0 0.4 1 0.6 0
	大	0 0 0 0 0 0 0.8
洗涤时间	短	1 0.5 0 0 0 0 0 0
	较短	0.4 0.8 1 0.8 0.4 0.2 0 0
	标准	0 0 0 0.2 0.6 1 0.6 0.2
	长	0 0 0 0 0 0.2 0.5 0.8

根据模糊规则，可得到网络的训练样本 P 和 T。完整的 MATLAB 代码如下：

```
%获得网络的输入样本 P 和目标样本 T
P=[1 0.6 0.1 0 0 0 0    1 0.5 0 0 0 0 0;
   1 0.6 0.1 0 0 0 0    0 0.5 1 0.4 0 0 0;
   1 0.6 0.1 0 0 0 0    0 0 0 0.4 1 0.6 0;
   1 0.6 0.1 0 0 0 0    0 0 0 0 0 0 0.8;
```

```
   0 0.6 0.6 0 0 0 0    1 0.5 0 0 0 0 0 0;
   0 0.6 0.6 0 0 0 0    0 0.5 1 0.4 0 0 0;
   0 0.6 0.6 0 0 0 0    0 0 0 0.4 1 0.6 0;
   0 0.6 0.6 0 0 0 0    0 0 0 0 0 0 0.8;
   0 0 0.6 1 0 0 0      1 0.5 0 0 0 0 0 0;
   0 0 0.6 1 0 0 0      0 0.5 1 0.4 0 0 0;
   0 0 0.6 1 0 0 0      0 0 0 0.4 1 0.6 0;
   0 0 0.6 1 0 0 0      0 0 0 0 0 0 0.8;
   0 0 0 0 1 0.6 0      1 0.5 0 0 0 0 0 0;
   0 0 0 0 1 0.6 0      0 0.5 1 0.4 0 0 0;
   0 0 0 0 1 0.6 0      0 0 0 0.4 1 0.6 0;
   0 0 0 0 1 0.6 0      0 0 0 0 0 0 0.8]';
T=[1 0.5 0 0 0 0 0 0;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0.4 0.8 1 0.8 0.4 0.2 0 0;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0 0 0.2 0.5 0.8;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0 0 0.2 0.5 0.8;
   0 0 0 0 0 0.2 0.5 0.8;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0.2 0.6 1 0.6 0.2;
   0 0 0 0 0 0.2 0.5 0.8;
   0 0 0 0 0 0.2 0.5 0.8]';
%根据 Kolmogorov 定理，由输入层有 14 个结点，所以中间层有 29 个结点
%中间层神经元的传递函数为 tansig
%输出层有 8 个结点，其神经元传递函数为 logsig
%训练函数采用 traingdx
net=newff(minmax(P),[29,8],{'tansig','logsig'},'traingdx');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
Y=sim(net,P);
%求训练值在每一个点上的误差
for i=1:16
    x(i)=norm(Y(:,i));
end
plot(1:16,x);
```

网络的训练结果为：

```
TRAINGDX, Epoch 0/1000, MSE 0.34484/0.001, Gradient 0.0932404/1e-006
```

```
TRAINGDX, Epoch 25/1000, MSE 0.34167/0.001, Gradient 0.0941182/1e-006
TRAINGDX, Epoch 50/1000, MSE 0.331398/0.001, Gradient 0.096924/1e-006
TRAINGDX, Epoch 75/1000, MSE 0.292917/0.001, Gradient 0.104688/1e-006
TRAINGDX, Epoch 100/1000, MSE 0.163575/0.001, Gradient 0.0782731/1e-006
TRAINGDX, Epoch 125/1000, MSE 0.0774625/0.001, Gradient 0.0208932/1e-006
TRAINGDX, Epoch 150/1000, MSE 0.0480671/0.001, Gradient 0.00789182/1e-006
TRAINGDX, Epoch 175/1000, MSE 0.0396982/0.001, Gradient 0.00460771/1e-006
TRAINGDX, Epoch 200/1000, MSE 0.00574098/0.001, Gradient 0.0104253/1e-006
TRAINGDX, Epoch 225/1000, MSE 0.00115385/0.001, Gradient 0.00177577/1e-006
TRAINGDX, Epoch 241/1000, MSE 0.00098956/0.001, Gradient 0.00236394/1e-006
TRAINGDX, Performance goal met.
```

网络的误差曲线如图 10-14 所示，由图可见，网络的最大误差不超过 0.2，说明网络性能是可以满足控制要求的。

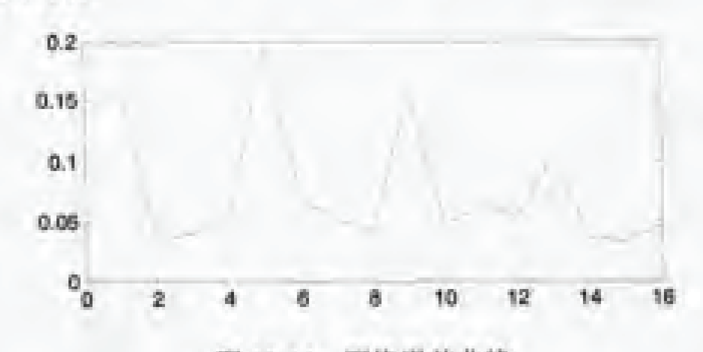


图 10-14 网络误差曲线

网络经过 241 次训练后，目标误差达到要求，结果如图 10-15 所示。

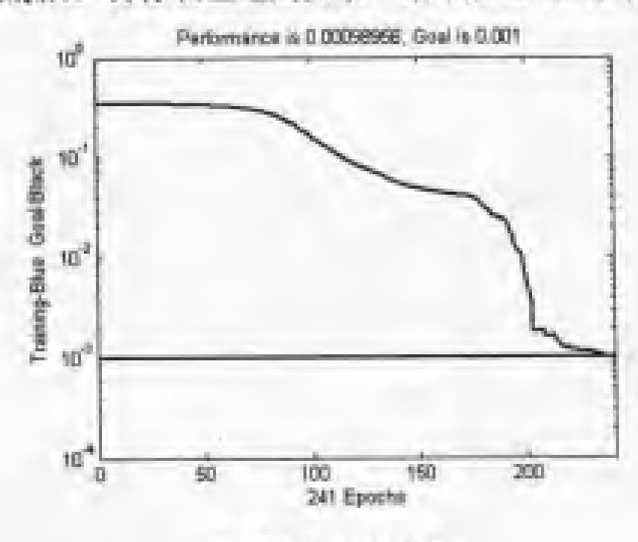


图 10-15 训练结果

10.4 小 结

本节研究了神经网络模糊控制的原理和神经模糊控制器的设计，并以神经模糊控制洗衣机为例，演示了神经网络在模糊控制中的应用。



总的说来，基于神经网络的模糊控制具有以下特点：

（1）可以直接从经验中获取知识，自动建立模糊规则和隶属函数。

（2）无需查表，节省内存空间，只需通过在线计算，便可得到控制器输出。

（3）具有较强的适应能力和联想能力，对于未出现过的样本，神经模糊控制器可通过记忆，联想产生合适的输出量对系统进行控制。



第 11 章　基于神经网络的自适应噪声抵消技术

本章通过大量的范例，全面介绍求解各种常见优化问题的标准算法，及其在 MATLAB 中的实现技术，帮助读者熟练使用 MATLAB 优化工具箱来解决工程实际问题。

本章主要内容包括：

- 自适应噪声抵消实现原理
- 噪声抵消系统的 MATLAB 仿真

11.1　引　　言

滤波理论与技术是信号处理的主要内容。所谓滤波，实际上是完成从输入到输出的某种映射关系。滤波的种类很多，最简单的滤波器是权系数固定的线性滤波器。为了提高滤波性能，滤波器的权系数（传递函数）往往随输入信号而发生变化，这种滤波称为自适应滤波（ANC），如果滤波器的输入/输出关系具有非线性映射特性，那么对应的滤波称为非线性滤波。自适应非线性滤波是当今信号处理的又一重要发展方向，它是针对线性处理的许多不足和限制而发展起来的，在不少场合，可获得比线性处理更加优越的性能，因此具有广泛的应用前景。

对于 ANC 系统来说，其基本要求和主要特点是，其两路输入噪声必须统计相关，它们可以是线性相关的。使用线性相关 ANC 系统的自适应滤波器是线性滤波器，在这种情况下，对于平衡的输入噪声，自适应滤波器的稳态性能近似于维纳滤波器，但是当两路输入噪声间是非线性相关时，使用线性的 ANC 系统就遇到了困难。如果线性自适应算法用线性变换逼近非线性变换，为了达到足够高的精度，用于估计的权系数向量的维数将会爆炸性增长，自适应滤波器的估计速度必然会大为降低，其性能大幅下降，难以达到实时应用目的。这时为了有效地进行噪声抵消，就必须采用自适应的非线性系统做滤波器。为了克服自适应线性滤波器的缺点，人们研究了几种非线性自适应滤波器，常用的非线性滤波形式有 Volterra 型，虽然 Volterra 滤波器可以完成非线性映射，但由于在 Volterra 展开处理中，参量的数目随展开阶数指数性地增加，因此处理是很繁杂、困难的。

自 1986 年开始，神经网络随着反向传播（BP）算法的出现而得到复兴，此后神经网络的研究再度掀起高潮，由于神经网络具有高度并行性、很强的非线性、变换能力和学习能力，因此具有巨大的应用潜力，也为自适应非线性滤波提供了一种全新的思路和方法。

神经网络是模仿和延伸人脑智能、思维、意识等功能的非线性自适应系统，它具有全新的信息表达方式、高度平行分布处理、联想、自学习和自组织能力，以及极强的非线性

映射能力，这为它提供了巨大应用潜力，也为自适应非线性滤波提供了一种全新方法，激活函数为 Sigmoid 的多层前向神经网络能够以任意精度实现非线性函数逼近，所以可用 BP 神经网络组成非线性自适应滤波器。通过将滤波器的误差信号与神经网络中能量函数对应起来，可实现神经网络自适应滤波器。

11.2 自适应噪声抵消实现原理

噪声消除是信号处理的核心问题之一，通常实现最优滤波的滤波器为维纳滤波器与卡尔曼滤波器。它们均要求已知信号和噪声的先验知识，但在许多实际应用中往往无法预先得知，为此发展了自适应滤波器。1965 年美国斯坦福大学建成了第一个自适应噪声抵消（ANC）系统，随着计算机技术与集成电路技术的进步，新的自适应算法不断涌现出来，自适应噪声抵消在理论和应用上都得到了很大发展。自适应噪声抵消技术是一种能够很好地消除背景噪声影响的信号处理技术，应用自适应噪声抵消技术，可在未知外界干扰源特征、传递途径不断变化，以及背景噪声和被测对象声波相似的情况下，能够有效地消除外界声源的干扰获得高信噪比的对象信号，这一技术可为机械元件的噪声、振动等动态信号在测试环境不太理想的工作现场做测试分析和故障诊断时，提供有效的方法和依据，具有一定的理论意义和应用价值。

11.2.1 自适应滤波器

自适应滤波器自从 20 世纪 60 年代出现后，其理论在不断地发展与完善，应用也越来越广泛。自适应数字滤波器的原理如图 11-1 所示。图中：

$x(j)$——表示 j 时刻的输入信号值；

$y(j)$——表示 j 时刻的输出信号值；

$d(j)$——表示 j 时刻的参考信号值或所期望响应的信号值；

$e(j)$——误差信号，$e(j)=d(j)-y(j)$。

自适应数字滤波器的滤波参数受误差信号 $e(j)$的控制，根据 $e(j)$值而自动调整，使之适合下一时刻的输入 $x(j+1)$，以便使输出 $y(j+1)$接近于所期望的参考信号 $d(j+1)$。

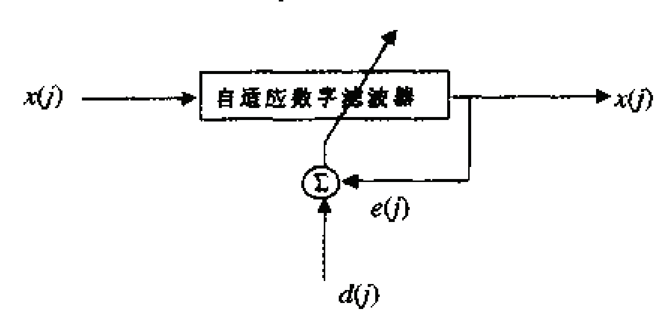


图 11-1　自适应数字滤波器原理

11.2.2 自适应噪声抵消系统基本原理

自适应噪声抵消系统除了需要原始输入外，还需要一个参考输入，供给与原始输入相关的噪声，以便抵消原始输入中的噪声，而对其中的有用信号几乎不产生什么畸变。

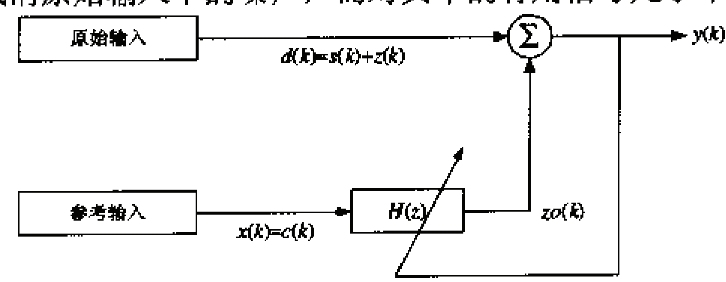


图 11-2 自适应噪声抵消系统

图 11-2 描述的是一典型的自适应噪声抵消系统，其中原始输入信号 $d(k)$ 是有用信号 $s(k)$ 与噪声 $z(k)$ 之和，参考输入信号 $x(k)$ 是与 $z(k)$ 相关的噪声 $c(k)$，假设 $s(k)$、$z(k)$ 与 $c(k)$ 是零均值的平稳随机过程，$s(k)$ 与 $z(k)$、$c(k)$ 不相关，由图 11-2 可见，自适应滤波器的输出 $zo(k)$ 为 $c(k)$ 的滤波信号，因此，自适应噪声抵消系统的输出 $y(k)$ 为：

$$y(k)=s(k)+z(k)-zo(k)$$

而

$$y^2(k)=s^2(k)+[z(k)-zo(k)]^2+2s(k)[z(k)-zo(k)]$$

根据前面不相关的假定，对上式两边取数学期望，可得：

$$E[y^2(k)]=E[s^2(k)]+E[(z(k)-zo(k))^2]$$

信号功率 $E[s^2(k)]$ 与自适应滤波器的调节无关，因此，自适应滤波器调节使 $E[y^2(k)]$ 最小，也就是 $E[(z(k)-zo(k))^2]$ 最小，$E[(y(k)-s(k))^2]$ 也最小，即自适应噪声抵消系统的输出信号 $y(k)$ 与有用信号 $s(k)$ 的均方差最小。

在理想情况下，$zo(k)=z(k)$，则 $y(k)=s(k)$，这时，自适应滤波器自动调节其脉冲响应，将 $c(k)$ 加工成 $z(k)$，与原始输入信号 $d(k)$ 中的 $z(k)$ 相减，使输出信号 $y(k)$ 由于噪声完全被抵消，而等于有用信号 $s(k)$。

可以证明，自适应滤波器能完成上述任务的必要条件为：参考输入信号 $x(k)=c(k)$ 必须与被抵消的信号（现为噪声）$z(k)$ 相关。

一个实际的噪声抵消系统的情况比如图 11-1 所示的还要复杂一些，这是因为输入还可能有一些独立噪声源，即与参考输入无关的噪声及干扰，如图 11-3 所示。



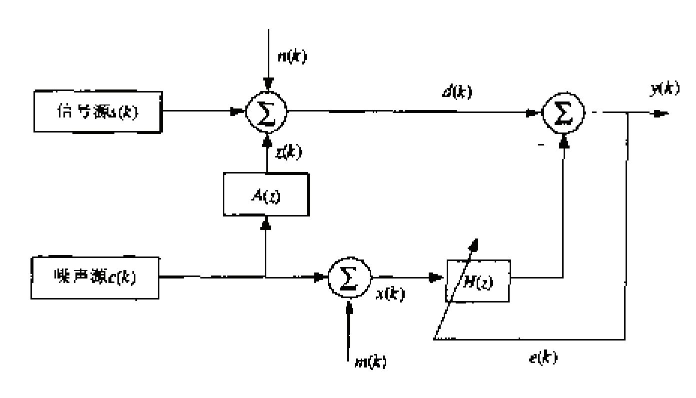


图 11-3 常见的自适应噪声抵消系统

11.3 噪声抵消系统的 MATLAB 仿真

11.3.1 BP 网络模型建立

根据分析结论，这里构造一个 1-4-1 型 BP 神经网络，隐含层结点取 4 个，如图 11-4 所示。

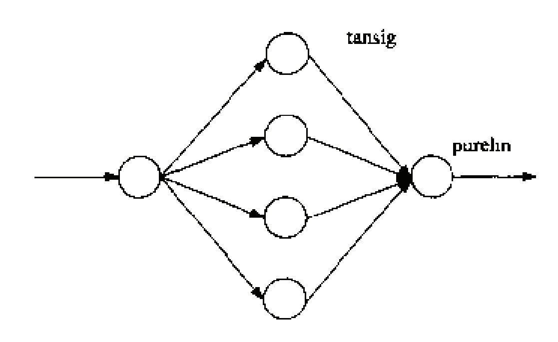


图 11-4 BP 神经网络模型

本网络中的隐含层变换函数为 tansig，它可以将神经元的输入范围 $(-\infty,+\infty)$ 映射到 $(-1,+1)$，是可微函数，非常适用于利用 BP 训练的神经元。如果 BP 网络的最后一层是 Sigmoid 型神经元，那么整个网络的输出就会限制在一个较小的范围内；如果是 purelin 型线性神经元，则整个网络的输出可为任意值。

11.3.2 基于神经网络工具箱的 BP 网络学习和训练

神经网络工具箱为训练神经网络提供了帮助，可以利用它提供的函数对网络进行初始化、仿真和训练，并通过变化的图形观察其动态训练过程。

如图 11-5 所示，给出了 BP 神经网络的训练过程。

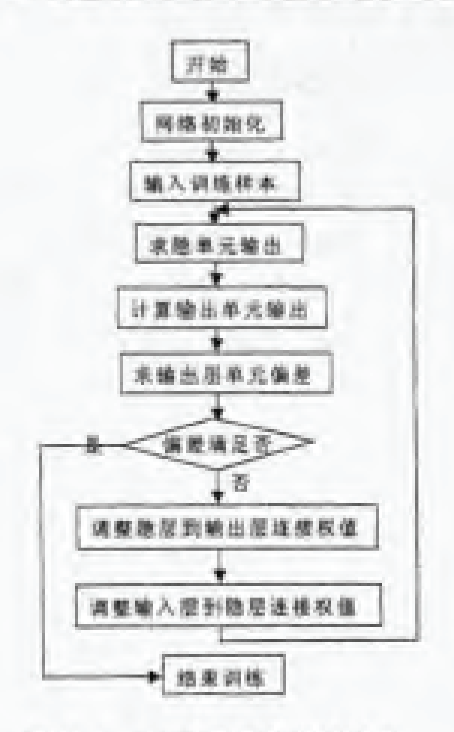


图 11-5 BP 神经网络训练过程

以下是 MATLAB 的实现代码：

```
% 定义样本
t=0:0.01:10;
y=randn(size(t));
yn=sin(y);
p=y(1:30)
t=yn(1:30)

% 初始化 BP 网络
%其中，p 为输入量，隐层结点数为 4，输出层结点数为 1
% tansig 和 purelin 分别为隐层和输出层的变换函数
%这样可得隐层和输出层的初始权值和阈值 w1，b1，w2，b2
[w1,b1,w2,b2]=initff(p,4,'tansig',1,'Purelin');

% BP 网络的训练
%利用 BP 学习规则训练前向网络，使其完成函数逼近、矢量分类和模式识别，选择训练参数，
%并指示如何进行训练
%一旦训练达到最大的训练次数或网络误差平方和降低到误差之下，都会使网络停止学习
tp(1)=10;  %指定两次更新显示间的训练次数
tp(2)=1000; %指定训练的最大次数
tp(3)=0.01; %误差平方和指标
tp(4)=0.01; %指定学习速率，即权值和阈值更新的比例
tp=[tp(1) tp(2) tp(3) tp(4)];
[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'tansig',w2,b2,'Purelin',p,t,tp);
% 画出误差变化曲线
figure(1);
plot([0:1:te],tr);
```

```
% 计算网络仿真输出
a=simuff(p,w1,b1,'tansig',w2,b2,'Purelin');
figure(2);
plot([0:0.01:2.9],t,'+',[0:0.01:2.9],a,'-');
```

在这段程序中，BP 网络训练时，函数利用单层权值矢量 w、阈值量 b 及转移函数成批训练网络，使当输入 p 时，网络的输出为目标矢量 t，在这个环节将得到新的权值矢量 w 和阈值 b、训练的实际迭代次数 te，以及记录网络训练过程的误差平方和行矩量 tr。

误差变化曲线如图 11-6 所示，函数逼近曲线如图 11-7 所示。

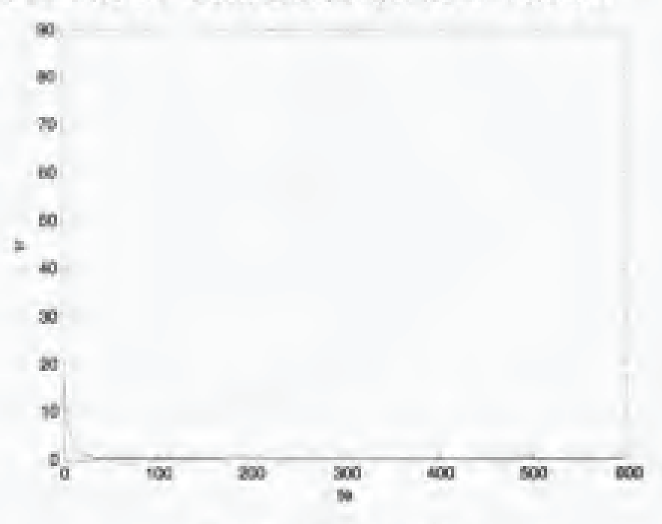


图 11-6　误差变化曲线

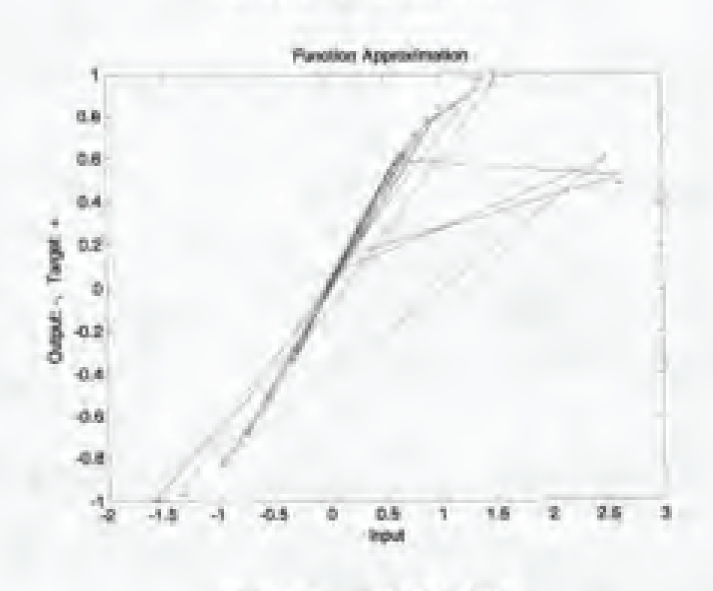


图 11-7　函数逼近曲线

根据误差曲线的变化，可以看到经过 600 多次训练之后，它最终逼近于 0.01，从函数逼近曲线可看出输出曲线非常逼近于目标曲线，证明训练后的网络具有较好的拟合性，在本环节中训练了网络并得到了构造 BP 网络的基本要素权值和阈值。

11.3.3 基于 Simulink 的噪声抵消系统设计及动态仿真

如图 11-8 所示，是我们针对噪声抵消系统，建立的 Simulink 仿真模型。可以发现，使用 Simulink，能使系统模块化，工作量大大减少。

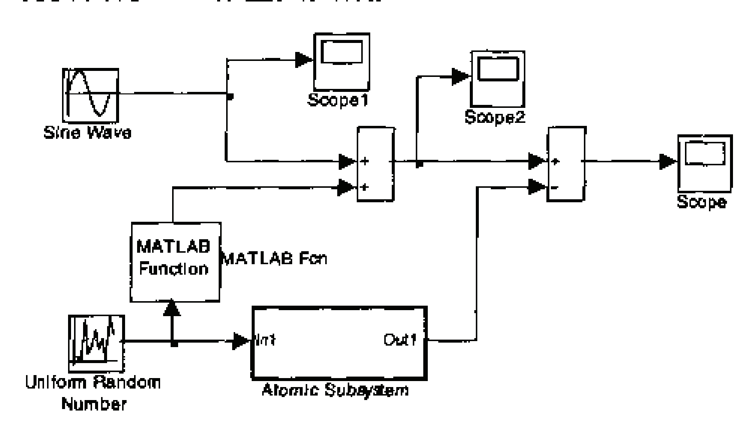


图 11-8 噪声抵消系统 Simulink 仿真模型

这里有个使用技巧。在系统建模与仿真中，会经常遇到很复杂的系统结构，难以用一个单一的模型框图进行描述。通常地，需要将这样的框图分解成若干个具有独立功能的子系统，Simulink 支持这样的子系统结构。此外，还可以将一些常用的子系统封装成为一些模块，这些模块的用法也类似于标准的 Simulink 模块，更进一步地还可以将自己开发的一系列模块做成自己的模块组或模块集。

用通俗的话说，就是在 Simulink 主环境下构造大型控制系统方框图，如果把系统中的每个元件都显示出来，就会遇到空间不够的问题，往往整整一屏都放不下系统方框图的某个局部。解决这个问题的办法就是将实现某一功能的各个元件组合封装起来，形成一个子系统（Subsystem）。如图 11-9 所示的子模块图就是图 11-8 中的 Atomic Subsystem 封装的详细实现。如果用鼠标双击 Subsystem 模块，就会显示其基本模型。

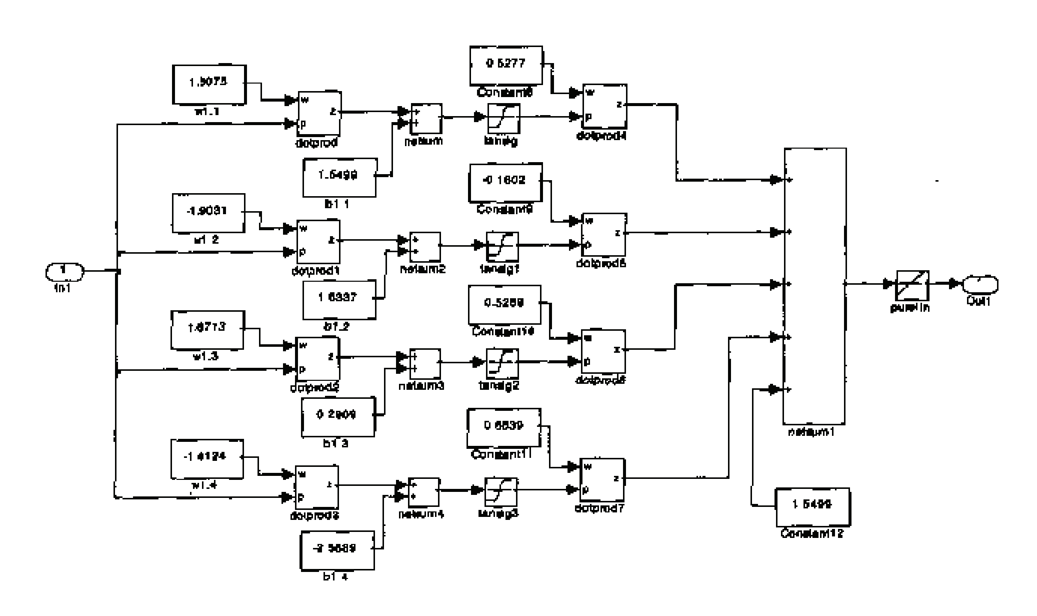


图 11-9 BP 网络构造子模块图

在建立了 Simulink 模型之后，我们就可以得到下面的结果。如图 11-10 所示是原始 sine 信号，如图 11-11 所示是被随机噪声污染以后的信号，而如图 11-12 所示是经过噪声抵消后的信号。可以发现，经过神经网络抵消后，噪声得到了明显的抑制，说明我们介绍的神经网络噪声抵消方法是非常有效、实用的。

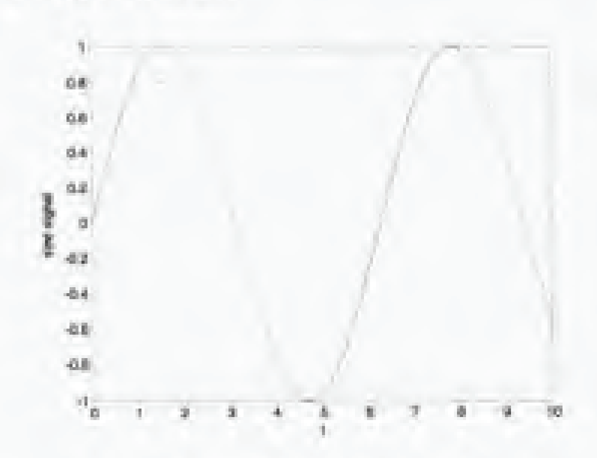


图 11-10　原始 sine 信号

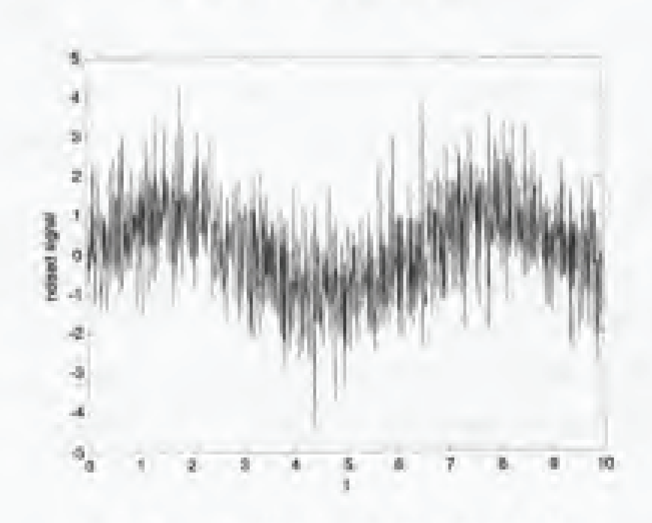


图 11-11　被随机噪声污染以后的信号

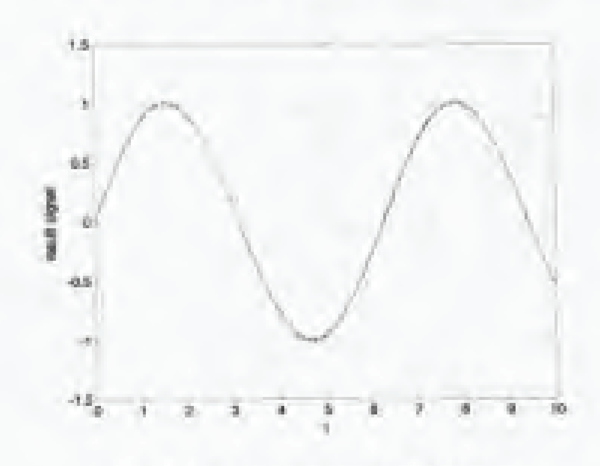


图 11-12　经过噪声抵消后的信号

11.4 小　　结

自适应噪声抵消技术是一种能够很好地消除背景噪声影响的信号处理技术，可为机械元件的噪声、振动等动态信号在测试环境不太理想的工作现场作测试分析和故障诊断提供了有效的方法和依据。

ANC 系统的基本要求和主要特点是其两路输入噪声必须统计相关，但是当两路输入噪声间是非线性相关时，使用线性的 ANC 系统就遇到了困难。这时为了有效地进行噪声抵消，就必须采用自适应的非线性系统做滤波器。神经网络是模仿和延伸人脑智能、思维、意识等功能的非线性自适应系统，激活函数为 Sigmoid 的多层前向神经网络能够以任意精度实现非线性函数逼近，所以可用 BP 神经网络组成非线性自适应滤波器。

仿真结果表明基于 BP 神经网络的噪声抵消能有效地清除噪声，具有很好的鲁棒性能。感兴趣的读者可以考虑以下几个问题：(1) 反向传播算法的收敛速度比较慢，一般需要成千上万次的迭代计算；(2) 存在能量函数局部极小值；(3) 网络隐含层神经元数目的选取和连接权 W 初值的选取，往往要靠经验，针对上述问题，下面给出一些对反向传播算法的改进方法。

1．改进算法的优化计算方法

$$\Delta W^{(n)} = -\eta \frac{\partial e_p}{\partial w} + \alpha \Delta W^{(n-1)}$$

式中：

$\Delta W^{(n)}$——第 n 次迭代计算时连接权的修正值；

$\Delta W^{(n-1)}$——前一次迭代计算时所得的连接权修正值；

a——动量因子，一般取接近 1 的数。

据有关文献介绍，取 $\eta = 0.2 \sim 0.5$，$a = 0.9 \sim 0.98$ 时，算法的收敛速度较快。

确定连接权修正值 ΔW 的计算过程，实际上就是优化计算中的梯度下降法。当能量函数对应的误差曲面为窄长型时，这种算法在谷的两壁跳来跳去，影响了网络的收敛速度。对算法最普通的改进方法是增加附加动量项。

2．改进激发函数

在网络的隐含层中，神经元的激发函数一般选取形式为 Sigmoid 函数：

$$O_i = \frac{1}{1+e^{-b}}$$

为了进一步提高神经网络的实用性，将 S 型函数改进为如下形式：

$$O_i = \frac{1}{a+be^{-d}}$$

式中，a、b、c 为可调参数，通过改变系数 a、b 和 c 的值，改变了 S 型函数图像的几何形状，也就是改变了相应神经元的输入/输出特性。

3．调整网络的结构

反向传播算法对网络的结构很敏感。一般网络的结构是根据经验设定的，如果在网络的训练过程中，对网络的结构进行调整，则可以取得较好的结果，此外，连接权的初始值对网络的学习过程也有很大的影响，一般的做法是选取不同的连接权初始值对网络进行训练，选取精度最高的网络供实际使用。

参 考 文 献

[1] 张铃，张钹．人工神经网络理论及应用[M]．浙江：浙江科学技术出版社，1997

[2] Neural Network Toolbox．MathWorks，2004

[3] 李学桥．神经网络工程应用[M]．重庆：重庆大学出版社，1995

[4] 沈世镒．神经网络系统理论及其应用[M]．北京：科学出版社，1998

[5] 张定会，邵惠鹏．基于神经网络的故障诊断推理方法[J]．上海交通大学学报，1999，5：619～621

[6] 沈清，胡德文，时春。神经网络应用技术[M]．长沙：国防科技大学出版社，1993

[7] 陈一梅，徐造林．基于神经网络的河道浅滩演变预测模型[J]．水利学报，2002.8

[8] 张捷．基于神经网络的齿轮箱智能故障诊断技术的研究[硕士学位论文]．江苏大学，2003.6

[9] 莫剑冬，徐章遂，米东．应用概率神经网络诊断自行火炮发动机的故障[J]．华北工学院测试技术学报，Vol.14，No.1，2001

[10] 黄加亮．RBF 神经网络在船用低速柴油机故障诊断中的应用研究[硕士学位论文]．大连海事大学，2000.3

[11] 赵闯，刘凯，李电生．基于广义回归神经网络的货运量预测[J]．铁道学报，Vol.26，No.1，2004

[12] 高宁．基于 BP 神经网络的农作物虫情预测预报及其 MATLAB 实现[硕士学位论文]．安徽农业大学，2003.5

[13] 夏道明．基于神经网络的自适应噪声抵消的研究[硕士学位论文]．武汉理工大学，2002.3

[14] 付强，王志良，梁川．自组织竞争人工神经网络在土壤分类中的应用[J]．水土保持通报，Vol.22，No.1，2002

[15] The MathWorks．Inc. http://www.mathworks.com，2004

[16] 杨建强，罗先香．地下水动态预测的径向基函数法[J]．工程勘察，2001.5

[17] 闻新等．MATLAB 神经网络仿真与应用[M]．科学出版社，2003.7

[18] 张建业，李学仁，陆阿坤．BAM 网络在飞机空中特情处置中的应用[J]．空军工程大学学报（自然科学版），Vol.3，No.6，2002.12